Lusheng Wang (Ed.)

LNCS 3595

Computing and Combinatorics

11th Annual International Conference, COCOON 2005
Kunming, China, August 2005
Proceedings

Lecture Notes in Computer Science 3595

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Lusheng Wang (Ed.)

Computing and Combinatorics

11th Annual International Conference, COCOON 2005
Kunming, China, August 16-19, 2005
Proceedings

Volume Editor

Lusheng Wang
City University of Hong Kong
Department of Computer Science
83 Tat Chee Ave., Kowloon, Hong Kong, China
E-mail: lwang@cs.cityu.edu.hk

Library of Congress Control Number: 2005929995

CR Subject Classification (1998): F.2, G.2.1-2, I.3.5, C.2.3-4, E.1, E.5, E.4

ISSN 0302-9743
ISBN-10 3-540-28061-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-28061-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11533719 06/3142 5 4 3 2 1 0

Preface

The papers in this volume were presented at the *Eleventh Annual International Computing and Combinatorics Conference* (COCOON 2005), held August 16–19, 2005, in Kunming, China. The topics cover most aspects of theoretical computer science and combinatorics related to computing.

Submissions to the conference this year were conducted electronically. A total of 353 papers were submitted, of which 96 were accepted. So the competition is very fierce. The papers were evaluated by an international program committee consisting of Tatsuya Akutsu, Vineet Bafna, Zhi-Zhong Chen, Siu-Wing Cheng, Francis Chin, Sunghee Choi, Bhaskar DasGupta, Qizhi Fang, Martin Farach-Colton, Raffaele Giancarlo, Mordecai Golin, Peter Hammer, Tsan-sheng Hsu, Sorin C. Istrail, Samir Khuller, Michael A. Langston, Jianping Li, Weifa Liang, Guohui Lin, Bernard Mans, Satoru Miyano, C. K. Poon, R. Ravi, David Sankoff, Shang-Hua Teng, H. F. Ting, Seinosuke Toda, Takeshi Tokuyama, Peng-Jun Wan, Lusheng Wang, Todd Wareham, Jinhui Xu, Xizhong Zheng, Kaizhong Zhang and Binhai Zhu.

The authors of submitted papers came from more than 25 countries and regions. In addition to the selected papers, the conference also included three invited presentations by Alberto Apostolico, Shang-Hua Teng, and Leslie G. Valiant. This year's Wang Hao Award (for young researchers) was given to the paper *Approximating the Longest Cycle Problem on Graphs with Bounded Degree* by Guantao Chen, Zhicheng Gao, Xingxing Yu and Wenan Zang.

I would like to thank all the people who made this meeting possible and enjoyable: the authors for submitting papers and the program committee members and external referees for their excellent work. I would also like to thank the three invited speakers and the local organizers and colleagues for their assistance.

August 2005 Lusheng Wang

Organization

COCOON 2005 was sponsored by the Academy of Mathematics and System Sciences of the Chinese Academy of Sciences.

Program Committee Chair

Lusheng Wang (City University of Hong Kong, Hong Kong, China)

Program Committee

Tatsuya Akutsu (Kyoto University, Japan)
Vineet Bafna (University of California, San Diego, USA)
Zhi-Zhong Chen (Tokyo Denki University, Japan)
Siu-Wing Cheng (The Hong Kong University of Science and Technology, China)
Francis Chin (The University of Hong Kong, China)
Sunghee Choi (KAIST, Korea)
Bhaskar DasGupta (University of Illinois at Chicago, USA)
Qizhi Fang (Ocean University of China, China)
Martin Farach-Colton (Rutgers University, USA)
Raffaele Giancarlo (University of Palermo, Italy)
Mordecai Golin (Hong Kong Univ. of Science and Technology, China)
Peter Hammer (Rutgers University, USA)
Tsan-sheng Hsu (Academia Sinica, Taiwan)
Sorin C. Istrail (Celera, USA)
Samir Khuller (University of Maryland, USA)
Michael A. Langston (University of Tennessee, USA)
Jianping Li (Yunnan University, China)
Weifa Liang (The Australian National University, Australia)
Guohui Lin (University of Alberta, Canada)
Bernard Mans (Macquarie University, Australia)
Satoru Miyano (The University of Tokyo, Japan)
C. K. Poon (City University of Hong Kong, China)
R. Ravi (Carnegie Mellon University, USA)
David Sankoff (University of Ottawa, Canada)
Shanghua Teng (Boston University, USA)
H. F. Ting (The University of Hong Kong, China)
Seinosuke Toda (Nihon University, Japan)
Takeshi Tokuyama (Tohoku University, Japan)
Peng-Jun Wan (City University of Hong Kong, China)
Todd Wareham (Memorial University of Newfoundland, Canada)
Jinhui Xu (State University of New York, USA)

Xizhong Zheng (Brandenburg University of Technology Cottbus, Germany)
Kaizhong Zhang (University of Western Ontario, Canada)
Binhai Zhu (Montana State University, USA)

Organizing Committee

Xiaodong Hu (Chinese Academy of Sciences, China) (**Chair**)
Degang Liu (Chinese Academy of Sciences, China)
Jie Hu (Chinese Academy of Sciences, China)

Referees

Dimitris Achilioptas
Hee-Kap Ahn
Tetsuo Asano
Martin Baca
Tanya Berger-Wolf
Arijit Bishnu
Rhonda Chaytor
Shihyen Chen
Dave Churchill
Michael Elkin
Thomas Erlebach
Rohan Fernandes
Amos Fiat
Stanley Fung
Matthew Hamilton
Rafi Hassin
Yoo-Ah Kim
P. Dwight Kuo
Yan Li
Tien-Ching Lin
Bruce Litow
Azarakhsh Malekian
Miguel Mostiero
Hyeon-Suk Na
Sheung-Hung Poon
Kirk Pruhs
Balaji Raghavachari
Kenneth W. Regan
Robert Rettinger
Xiaojun Shen
Igor Shparlinski
Aravind Srinivasan
Ron Steinfeld
Hisao Tamaki
Tatsuie Tsukiji
Caoan Wang
Yajun Wang
Chuan-Kun Wu
Xiaodong Wu
Huaming Zhang
Yan Zhang

Table of Contents

Invited Lectures

Bioinformatics

Networks

String Algorithms

Scheduling

Complexity

Steiner Trees

Graph Drawing and Layout Design

Quantum Computing

Randomized Algorithms

Geometry

Codes

Finance

Facility Location

Graph Theory

Graph Algorithms

Others

Completeness for Parity Problems*

Leslie G. Valiant

Division of Engineering and Applied Sciences, Harvard University
Cambridge, MA 02138, USA

Abstract. In this talk we shall review recent work on holographic algorithms and circuits. This work can be interpreted as offering formulations of the question of whether computations within such complexity classes as NP, $\oplus$P, BQP, or #P, can be efficiently computed classically using linear algebra. The central part of the theory is the consideration of gadgets that map simple combinatorial constraints into gates, assemblies of which can be evaluated efficiently using linear algebra. The combinatorial constraints that appear most fruitful to investigate are the simplest ones that correspond to problems complete in these complexity classes. With this motivation we shall in this note consider the parity class $\oplus$P for which our understanding of complete problems is particularly limited. For example, among the numerous search problems for which the existence of solutions can be determined in P and the counting problem is known to be #P-complete, the #P-completeness proof does not generally translate to a $\oplus$P-completeness proof. We observe that in one case it does, and enumerate several natural problems for which the complexity of parity is currently unresolved. We go on to consider two examples of NP-complete problems for which $\oplus$P-completeness can be proved but is not immediate: Hamiltonian circuits for planar degree three graphs, and satisfiability of read-twice Boolean formulae.

1 Introduction

The class $\oplus$P is the class of sets S such that there is a polynomial time nondeterministic Turing machine that on input $x \in S$ has an odd number of accepting computations, and on input x $\notin S$ has an even number of accepting computations ([V79], [PZ83], [GP86]). It formalizes the question of the parity of the number of solutions to combinatorial problems. It is known that $\oplus$P has at least the computational power of NP, since NP is reducible to $\oplus$P via (one-sided) randomized reduction [VV86]. Also, the polynomial hierarchy is reducible to it via two sided randomized reductions [TO92]. The class FewP of sets for which there exist NP machines with few accepting computations is a subclass of it [CH90]. Further, there exist decision problems, such as graph isomorphism, that are not known to be in P but are known to be in $\oplus$P [AK02]. The class $\oplus$P has been related to other complexity classes via relativization [BBF98].

* This research was supported in part by grants NSF-CCR-03-10882, NSF-CCR-98-77049, and NSF-CCF-04-27129.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 1–8, 2005.

2 Some Easily Computed Parity Problems

There are several problems for which counting the number of solutions is #P-complete while computing the parity, i.e. whether there is an odd or even number of solutions, is polynomial time computable. The prime example is that of perfect matchings in bipartite graphs where exact counting corresponds to computing the permanent of a 0/1 matrix. The parity problem corresponds to computing the permanent modulo two, which is the same as the determinant modulo two, and is therefore computable in polynomial time via linear algebra computations. Many variants of this matching problem, such as those in which the matchings need not be perfect, or the graph bipartite, also have polynomially computable parity problems for similar reasons.

A further category of problems with polynomially computable parity is that of *read-twice formulae*. These are formulae where each variable occurs at most twice. In particular, we consider formulae that are *conjunctive* in the sense that they consist of conjunctions of clauses that each depend on at most three variables. In [V02] it is shown, by parity preserving reductions to matchings, that the parity of such read-twice formulae can be computed in P provided the clauses are (any mixture) of the following forms: a clause dependent on at most two variables, or a clause of any of the forms $xyz, x(y=z), x'yz+xy'z+xyz', x(y+z), x \oplus y \oplus z = 1, xy+yz+zx, x+(y=z), xy+(y=z)$, where each of these forms also allows any of x, y, z to be replaced by their negations x', y', or z'. (We note that with respect to the existence of solutions, a complete analysis of the relative expressivity of read-twice formulae composed of any one of these forms can be found in [CB05]).

Remarkably, there are a large number of natural parity problems for which we currently have no hint as to their complexity. In particular, almost any #P-complete problem for which existence of solutions is known to be in P, but not via matchings, is a potential such open problem. The following are notable examples:

(i) $\oplus$2SAT - the parity of the number of solutions of 2-CNF formulae.
(ii) The parity of the number of solutions of read-twice monotone formulae.
(iii) The parity of the number of solutions of read-twice 3-CNF formulae.

All three are open even if the formulae is restricted to be planar, and (ii) and (iii) are open if the formulae are not restricted to be read-twice.

3 Some $\oplus$P-Complete Problems

One can define $\oplus$P-completeness with respect to various reductions. In this paper we shall use the term in the sense of polynomial time many-one (Karp) reductions.

We first consider NP search problems for which the existence of solutions can be decided in polynomial time. There are numerous NP search problems for which the existence of solutions can be determined in P but counting their

number is #P-complete [V79]. For the vast majority of these, however, the #P-completeness reduction does not translate to a ⊕P-completeness proof. For example, the general technique of evaluation and interpolation requires an unbounded number of field elements for the interpolation process. For some other reductions such as for the permanent, even integer constants are needed, and the parity problem is, in fact, known to be polynomial time computable.

The one natural problem we know for which existence is in P but parity is ⊕P-complete concerns monotone formulae. If we denote by ⊕MonFormulaSat the problem of determining whether a monotone Boolean formula is satisfied by an odd or even number of solutions then we have:

Theorem 1. *⊕MonFormulaSat is ⊕P-complete.*

Proof. In [V79] it is shown that any CNF formula F over variables $\underline{x}$ can be written as a formula $J = G(\underline{y}, \underline{z}) \wedge \neg H(\underline{y}, \underline{z})$ where G and H are monotone formulae and $\underline{y}$, $\underline{z}$ are new variables, such that J has the same number of solutions as F. But the number $\#J$ of solutions to J is clearly the difference

$$\#G(\underline{y}, \underline{z}) - \#(G(\underline{y}, \underline{z}) \wedge H(\underline{y}, \underline{z}))$$

between the numbers of solutions of two monotone formulae. Hence if one could compute ⊕MonFormulaSat in polynomial time for these two monotone formulae, then by taking the difference one could compute $\oplus F$. □

We now consider problems where the existence of solutions is NP-complete. For the majority of natural such problems there are reductions mapping the solutions of any polynomial time nondeterministic computation one-to-one (i. e. parsimoniously) to the solutions of the given problem. These then are #P- and ⊕P-complete by virtue of these reductions. Here we shall give ⊕P-completeness proofs for two NP-complete problems for which such a proof was not available before. For the first we use an NP-completeness proof that is not parsimonious, but for which we can show that parity is preserved nevertheless. We shall denote the following problem by ⊕ReadTwiceOppositeSat:

Input: A Boolean formula F consisting of the connectives $\wedge$ and $\vee$, where each variable occurs once negated and once unnegated.
Output: The parity of the number of satisfying assignments of F.

We can prove the following by adapting the proof of the NP-completeness of the same problem due to Hunt and Stearns [HS90].

Theorem 2. *⊕ReadTwiceOppositeSat is ⊕P-complete.*

Proof. First it is well known that any 3-CNF formula can be reduced parsimoniously to a read-thrice formula by replacing each occurrence of a literal by a new variable, and for each old variable conjoining the formula with a 2-CNF cycle. For example if the five new variables for the old variable x are a, b, c, d, and e, which represent occurrences x, x, x', x', x respectively, then the cycle would be

$(a + b')(b + c)(c' + d)(d' + e')(e + a')$. Clearly every way of satisfying this cycle will have exactly a half of the literals true and hence exactly one literal in each conjunct true. Also, the satisfied literals will be either all the first ones in the clauses or all the second ones. Hence the a, b, c, d, and e can be satisfied only in a way consistent with the sequence x, x, x', x', x.

We now map this formula F to a formula F^* that is read-twice and such that the parities of the number of solutions of F and F^* are the same. For each variable a in F we do the following: The positive occurrence of a in a cycle we leave unchanged, the occurrence of a outside the cycle we replace by a companion variable a^*, and the negated occurrence of a' in the cycle we replace by the conjunction $a'a^{*\prime}$. Thus formula F fragment

$$(a + b + c)(a + b')(e + a')$$

we modify to F^* fragment

$$(a^* + b^* + c^*)(a + b'b^{*\prime})(e + a'a^{*\prime}).$$

Clearly for assignments in which for every variable z, the values of z and z^* are the same, exactly those assignments satisfy F^* that satisfy F also.

Now the only way of satisfying a cycle in F^* remains that of satisfying exactly one term in each clause, since the cycle has become only more constrained by the introduction of the starred variables. If $\underline{v}$ is a vector of n variable values for F, let $(\underline{v}, \underline{v}^*)$ be a vector of $2n$ variable values for F^*. We denote by $\underline{v}^* \leq \underline{v}$ the condition that any variable v_i, if $v_i = 0$ then $v_i^* = 0$. □

Claim: The vector $(\underline{v}\ \underline{v}^*)$ satisfies F^* if and only if (i) $\underline{v}^* \leq \underline{v}$, (ii) $\underline{v}$ satisfies F, and (iii) $\underline{v}^*$ satisfies all the noncycle clauses of F^*.
Proof of Claim: In the backward direction suppose (i) and (ii) hold. Then the cycles in F^* will be satisfied since the $\underline{v}^*$ variables will either equal the $\underline{v}$ variables, or will be negative. Hence if (iii) holds also, then $(\underline{v}, \underline{v}^*)$ will satisfy F^*. In the forward direction, if we assume that $(\underline{v}, \underline{v}^*)$ satisfies F^*, then (i) is necessary since otherwise the number of terms satisfied in the cycles will be fewer than the number of clauses. Also, if $(\underline{v}, \underline{v}^*)$ satifies F^* then $\underline{v}^*$ satisfies the noncycle clauses of F^*, and (iii) follows. But then $\underline{v}$ also satisfies the noncycle clauses since $\underline{v}^* \leq \underline{v}$. Further if $(\underline{v}, \underline{v}^*)$ satisfies F^* then $\underline{v}$ satisfies the cycle clauses of F, and hence $\underline{v}$ satisfies F.

It follows from the above claim that for each solution $\underline{v}$ of F the solutions $(\underline{v}, \underline{v}^*)$ of F^* are exactly those such that $\underline{v}^* \leq \underline{v}$ and $\underline{v}^*$ satisfies the noncycle clauses of F^*. If $\underline{v}$ satisfies two literals in i noncycle clauses, and three literals in j noncycle clauses in F, then there will be $3^i 7^j$ such choices of $\underline{v}^*$, which is an odd number. Hence we have a reduction from 3-SAT to read-twice formulae that preserves the parity of the numbers of solutions. □

As an aside we note that from the above construction the #P-completeness of the corresponding counting problem can be deduced as follows. From F one

constructs F_k to be the same formula, but with each clause repeated k times. The F_k^* will have $(3^i7^j)^k$ solutions whenever F^* had 3^i7^j solutions, and F just one. By counting the solutions of F_k^* for enough values of k one can recover the number of solutions of F by polynomial interpolation.

Next we consider the problem of Hamiltonian circuits in planar degree three graphs. This has an a long history for the case of 3-connected regular degree three graphs. In 1880 Tait conjectured that all such graphs are Hamiltonian [T1880]. After many years Tutte found a counterexample [T46]. Subsequently, Garey, Johnson and Tarjan [GJT76] showed that the question of whether such graphs had Hamiltonian cycles was, in fact, NP-complete. Their reduction from 3SAT was not parsimonious. Recently Liskiewicz, Ogihara and Toda [LOT03] were able to find a reduction from 3SAT that for all solutions produced the same (even) number of Hamiltonian circuits in the constructed graph, thereby showing that this problem was #P-complete. Both these completeness proofs used the Tutte gadget, a graph used by Tutte in his original counterexample. Tutte had also shown that in any regular degree three graph the number of Hamiltonian circuits through any edge is always even. From this it can be easily deduced that any gadget with the sought after traversal properties of a Tutte gadget, will always be traversed an even number of times through any pair of external nodes, and hence that no reduction based on them can lead to a $\oplus$P-completeness result. We therefore consider the slightly less constrained problem PlHamDeg3 in which nodes of degree two are also allowed:

Input: Undirected planar graph G with nodes all of degree two or three.
Output: The parity of the number of Hamiltonian circuits in G.

For this problem we can show that a parsimonious reduction from 3SAT is possible and therefore that it is $\oplus$P-complete.

Theorem 3. *There is a parsimonious reduction from 3SAT to PlHamDeg3.*

Proof. We follow the proof for the regular degree three case given by Liskiewicz, Ogihara and Toda, [LOT03], which itself elaborates on [GJT76]. First we define the problem #3SAT* as follows:

Input: A Boolean CNF formula F where one clause has one literal and each of the remainder exactly three literals, such that in any satisfying assignment to F in each 3-clause$(x+y+z)$ containing three literals x, y, z in that order, it is the case that either exactly one literal is true, or x and y are true and z is false.
Output: The number of satisfying assignments to F.

We first observe that there is a parsimonious reduction from the standard #3SAT to #3SAT*. To see this we note that $(x+y+z)$ is logically equivalent to the formula

$$\begin{aligned}
&\exists u(x+y+u')((x+y) \Rightarrow u)(u+z)\\
\equiv\ &\exists u(x+y+u')(x \Rightarrow u)(y \Rightarrow u)(u+z)\\
\equiv\ &\exists u(x+y+u')(x'+u)(y'+u)(u+z)\\
\equiv\ &\exists u \exists w(x+y+u')(x'+u+w')(y'+u+w')(u+z+w')(w)
\end{aligned}$$

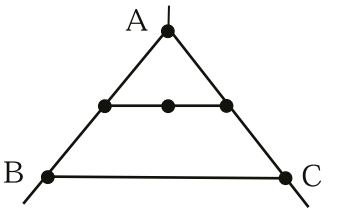


Fig. 1. A gadget such that any Hamiltonian circuit enters and leaves through {A,B} or {A,C}, but not {B,C}.

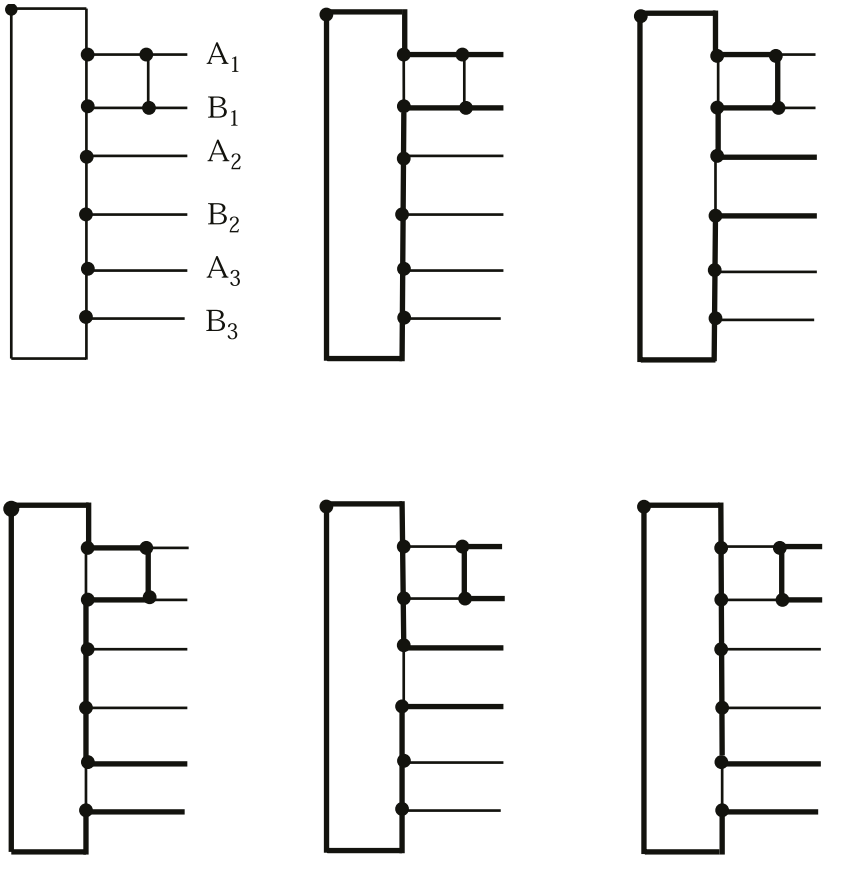


Fig. 2. Upper left figure shows OR-gadget. Remaining figures show the possible paths that can be part of a global Hamiltonian circuit. Note that the global construction is such that any Hamiltonian circuit entering at some A_i must next traverse B_i before it can traverse an A_j or B_j with $j \neq i$. Note also that for 3SAT* formulae the last of the cases shown never arises.

with the solutions mapping parsimoniously. Applying this construction to each 3-clause in the 3CNF formula, using a different u but the the same w for each

clause, will give the required construction. Then w' will be false in any satisfying assignment, as will u' in any clause in which either of the first two literals are true.

Next we construct the graph shown in Figure 1 that replaces the Tutte gadget used in [GJT76] and [LOT03]. It is clear that any Hamiltonian path will enter and leave either through {A, B} or {A, C}, but never through {B, C}. Further, unlike the Tutte gadget, there will be exactly one such transiting path in each of the two cases.

We then follow the construction of Liskiewicz, Ogihara and Toda [LOT03] that translates a CNF formula with one 1-clause and otherwise all 3-clauses to an undirected graph. We depart from their construction by using our Figure 1 graphs instead of Tutte gadgets, and using the graph shown in Figure 2 instead of their OR-gadgets. The special property of our Figure 2 gadget is that for each of the four allowed solutions of a clause $(x+y+z)$ in F, there will be exactly one state (as shown) that can be part of a Hamiltonian circuit in the overall graph. Further, there are no other states that can be part of such a global Hamiltonian circuit, exept for the one corresponding to $x = z = 1, y = 0$ which never arises.

We leave it to the reader to verify that this construction is a parsimonious mapping from 3SAT* to PlHamDeg3. □

The following is then an immediate consequence.

Corollary 1. *$\oplus$PlHamDeg3 is $\oplus$P-complete.* □

References

[AK02] V. Arvind and P.P. Kurur. Graph isomorphism is in SPP. Electronic*Colloquium on Computational Complexity*, Report 37, 2002.

[BBF98] R. Beigel, H. Buhrman, and L. Fortnow. NP might not be as easy as detecting single solutions, *STOC98*, ACM Press, pp.203-208 (1998).

[C71] S.A. Cook, The complexity of theorem proving procedures, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing* (1971), pp. 151–158.

[CB05] M. Cook, and J. Bruck, Implementability among predicates. *Technical Report.* California Institute of Technology, Pasadena, CA (2005).

[CH90] J.-Y. Cai and L. A. Hemachandra. On the power of parity polynomial time. *Math. Syst. Theory.* 23:2 (1990) 95-106.

[GJT76] M. R. Garey, D. S. Johnson and R. E. Tarjan, The Planar Hamiltonian Circuit Problem is NP-Complete, *SIAM J. Computing*, 5 (1976), pp. 704-714.

[GP86] L.M. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of Boolean functions. *Theoretical Computer Science*, 43 (1986) 43-58.

[HS90] H. B. Hunt III and R. E. Stearns. The complexity of very simple Boolean formulas with applications. *SIAM J. Comput.*, 19:1 (1990) 44-70.

[LOT03] M. Liskiewicz, M. Ogihara, and S. Toda, The Complexity of Counting Self-avoiding Walks in Subgraphs of Two-dimensional Grids and Hypercubes,*Theoretical Computer Science*, 304 (2003), 129-156

[PZ83] C. H. Papadimitriou, and S. Zachos.Two remarks on the power of counting. *Theoretical Computer Science*, (1983) 269-276.

[T1880] P.G. Tait, Remarks on coloring of maps. *Proc. Royal Soc.* Edinburgh 10 (1880) 729.

[TO92] S. Toda and M. Ogiwara, Counting classes are at least as hard as the polynomial time hierarchy. *SIAM J. Comput.*, 21:2 (1992) 316-328.

[T46] W. T. Tutte, On Hamiltonian circuits, *J. London Math. Soc.* 21, (1946), pp. 98-101.

[V79] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8 (1979) 189- 201.

[V02] L. G. Valiant, Quantum circuits that can be simulated classically in polynomial time, *SIAM J. on Computing*, 31:4 (2002) 1229-1254.

[V04] L. G. Valiant, Holographic algorithms, *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science*, Oct 17-19, (2004). IEEE Press, 306-315.

[V05] L. G. Valiant, Holographic circuits, Proc. 32nd ICALP, *LNCS*, Springer, (2005), to appear.

[VV86] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

Monotony and Surprise

Alberto Apostolico[1,2]

[1] Dipartimento di Ingegneria dell' Informazione
Università di Padova, Padova, Italy
[2] Department of Computer Sciences
Purdue University, West Lafayette, IN, USA
axa@cs.purdue.edu

Abstract. The extraction of recurrent patterns from sequences is a popular and intensive application, ubiquitous to many domains. Often, however, the process exposes more candidates than one can afford to inspect, thereby defying its whole purpose. While part of this problem is endemic, part of it can be attributed to the traditional definitions of what constitutes a pattern, that hinge alternatively on syntactic or statistical properties alone. It has been seen recently that this part of the problem may be mitigated by more prudent paradigms, in which the syntactic description of a pattern and the list of all of its occurrences are tightly intertwined. This approach leads to identify regions of monotonicity for some scores of surprise in use, within which it is enough to consider and weigh only extremal terms and values. This talk reviews concepts, constructs, and application results obtained along this line of research.

L. Wang (Ed.): COCOON 2005, LNCS 3595, p. 9, 2005.

Smoothed Analysis of Algorithms and Heuristics

Shang-Hua Teng

Boston University, Boston, MA 02215, USA
steng@cs.bu.edu
http://www.bu.edu/~steng

The theorists have long been challenged by the existence of remarkable algorithms and heuristics that are known by scientists and engineers to work well in practice, but whose theoretical analyses have been are negative or unconvincing. The root of the problem is that algorithms are usually analyzed in one of two ways: by worst-case or average-case analysis. The former can improperly suggest that an algorithm will perform poorly, while the latter can be unconvincing because the random inputs it considers may fail to resemble those encountered in practice.

We introduce smoothed analysis to help explain the success of some of these algorithms and heuristics. Smoothed analysis is a hybrid of worst-case and average-case analyses that inherits advantages of both. The smoothed complexity of an algorithm is the maximum over its inputs of the expected running time of the algorithm under slight random perturbations of that input, measured as a function of both the input length and the magnitude of the perturbations. If an algorithm has low smoothed complexity, then it should perform well on most inputs in every neighborhood of inputs.

In this talk, we will explain how smoothed analysis can help explain the excellent observed behavior of several algorithms of practical importance. We will survey progresses on applying smoothed analysis to the simplex method, Gaussian elimination, interior point methods, and some other optimization algorithms and heuristics. In particular, we show that the simplex algorithm has polynomial smoothed complexity. The simplex algorithm is the classic example of an algorithm that performs well in practice but takes exponential time in the worst case.

This is joint work with Daniel Spielman of MIT, and with John Dunagan (Microsoft Research) and Arvind Sankar (MIT).

References

[2004] Spielman D. A., Teng S.-H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. Journal of ACM. **51** (3) (2004) 385–463.

[2003] Spielman D. A., Teng S.-H.: Smoothed Analysis of Termination of Linear Programming Algorithms, Mathematical Programming, Series B, Vol 97, (2004) 275–404.

[2005] Sankar A., Spielman D. A., Teng S.-H. Smoothed Analysis of Condition Numbers and Growth Factors of Matrices, SIMAX, (2005 to apprear).

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 10–11, 2005.

[2005] Duangan J., Spielman D. A., Teng S.-H. Smoothed Analysis of Condition Numbers and Complexity Implications for Linear Programming. Mathematical Programming, Series A. (2005 to apprear)

[2004] Beier R., Vöcking B. Typical properties of winners and losers in discrete optimization. ACM STOC, (2004) 343–352.

[2003] Becchetti L., Leonardi S., Marchetti-Spaccamela A., Schafer G., Vredeveld T. Average Case and Smoothed Competitive Analysis of the Multi-Level Feedback Algorithm. IEEE FOCS (2003) 462-471.

Gene Network: Model, Dynamics and Simulation

Shiquan Wu and Xun Gu*

Center of Bioinformatics and Biological Statistics
Iowa State University, Ames, IA 50011, USA
sqwu@cs.iastate.edu, xgu@iastate.edu

Abstract. A gene network is modeled as a dynamical random graph whose vertices and edges represent genes and gene-gene interactions, respectively. The network grows through three biological mechanisms: (1) gene duplication and loss; (2) gene-gene interaction adding and removing; and (3) genome duplication. The evolutionary dynamics of gene networks is discussed. It is shown that: (1) the vertex degree distribution (i.e., the distribution of the number of the gene-gene interactions per gene) always follows power laws and the power law exponents may be changed by genome duplications; and (2) the network degree distribution (i.e., the distribution of the total number of the gene-gene interactions in the network) has a complex behavior: If no genome duplication occurs, it follows a power law. If a genome duplication occurs, it may be away from the power law state. However, after a sufficient long evolutionary time, it approaches to a power law tail. The dynamics is confirmed by computer simulations. By allowing genome duplications, our model and dynamics (describing the dynamic behavior of gene networks) are more realistic than other previous ones (containing only static behavior).

1 Introduction

Systems biology on genetic networks is an important area in bioinformatics. Its primary goal is to understand biological organisms at a system-level [1–3], because many cellular functions can only be understood by simultaneously studying a group of genes, proteins, and other biological components that interact each other. Various biological networks have been extensively studied (see [4] for a general review), including cellular functional links, regulatory pathways, signal transductions, protein interactions, and metabolic correlations [5–10].

Mathematically, the growth of a large complex network can be governed by two mechanisms [4, 11]: (1) the network grows continuously by adding new vertices and edges, and (2) new vertices connect preferentially to those already well connected. Under these assumptions, it has been shown that the degree distribution of every vertex in the network follows a power law [4, 12]; the power law exponent only depends on the network growth process, not the initial state of the network (e.g. mechanisms decide dynamics) [13]. It is also possible that the network diameter is small and the cluster coefficients of nodes are large [14–17].

* Corresponding author

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 12–21, 2005.

The duplications of both genes and genomes are the major resources for gene network growth and functional divergence [18]. Although several gene network models considered gene/genome duplications [6, 13, 19, 20], the mechanisms people introduced are somewhat artificial, due to mathematical convenience rather than biological reality [4, 10, 12, 13, 15, 16, 19–21]. Therefore, it is very interesting and important to develop gene networks allowing duplications for both genes and genomes under more biological realistic frameworks. This motivates our study.

In this paper, a gene network is formulated as a dynamical random graph whose vertices denote genes and whose edges represent gene-gene interactions among the genes. During evolution, the network grows through three biological mechanisms: (1) gene duplication and loss, (2) edge (i.e., gene-gene interaction) adding and removing, and (3) genome duplication. Moreover, preferential attachment is applied, i.e. a gene network is scale-free.

Based on these mechanisms, we study the network dynamics, focusing on the vertex degree distribution (i.e., the distribution of the gene-gene interactions per gene, which is a local statistical property in the netwotk) and the network degree distribution (i.e., the distribution of the gene-gene interactions in the network, which is a global statistical property in the netwotk). We show that the vertex degree distribution always follows power laws with different exponents, which can be changed by genome duplications. A genome duplication drives the network degree distribution away from its power law state. However, after a sufficient long time, the network degree distribution will somehow gradually recover to its power law state. The dynamics is verified by computer simulations. The genome duplication we allow makes our model and dynamics more general and realistic than the previous ones [2, 4, 13], which contain static behavior, whereas ours describe the dynamic behavior of gene networks. Our model and simulation also provide an efficient way to explore large scale complex biological networks.

2 Mathematical Model of Gene Network

A gene network is defined as a dynamical random graph G whose vertices represent genes and edges stand for gene-gene interactions. The network grows through three mechanisms: (1) gene duplication and loss; (2) gene-gene interaction adding and removing; and (3) genome duplication. A set of rates control gene duplication, gene loss, edge adding, edge removing, and genome duplication. Denote the graph

$$G = (V, E; g_d, g_l; \lambda, \mu; G_d, G_l).$$

The terms of the gene network G are defined as follows:

(1) V is the set of all vertices which represent genes (other biological components can be included, but here we restrict the discussion on genes).

(2) E is the set of all edges which stand for the gene-gene interactions (or influences) among the genes. For any two genes, if there exists any gene-gene interaction between them, the two genes are joined by an edge.

Initially, $V = V_0$ and $E = E_0$, forming a simple network, see Fig.1(a). Starting from the initial state ($V_0 = \{g_1, g_2, g_3, g_4\}$ and $E_0 = \{g_1g_3\}$), the vertices and edges are randomly changed due to the duplications of both genes and genomes.

(3) g_d is the rate of a gene duplication.

(4) g_l is the rate of a gene loss.

Any gene can be duplicated with the rate g_d. When a gene duplication occurs, a new duplicated copy of the gene is created. The duplicated gene at first inherits all gene-gene interactions from its original (i.e. parent) gene, which means that the new duplicated vertex is joined to all vertices connecting to the original vertex. The new duplicated vertex is also joined to its original vertex (see Fig.1(b), g_1^c is duplicated from g_1 and then joined to g_1). The duplicated gene can be lost (i.e., the vertex is removed) with the rate g_l during evolution.

(5) λ is the rate of edge adding for a new duplicated vertex to be joined to other unconnected vertex in the network.

(6) μ is the rate of edge removing for an existing edge to be removed.

The duplicated gene may undergo a series of edge adding and edge removing if it is not lost. Its existing edges are removed with the given rate μ, see Fig.1(c). When an edge is removed, it means that the corresponding gene-gene interaction is eliminated, i.e. its corresponding functions are unnecessary and disappeared.

(7) G_d denotes the rate of a genome duplication.

(8) G_l is the rate that a gene in the duplicated genome is lost or removed.

When a genome duplication occurs, all genes together with all of their interactions (i.e. edges) are duplicated. In addition, each duplicated gene is joined to its original gene by a new edge, see Fig.1(d). After a genome duplication, each duplicated gene in the new duplicated genome is removed with the rate G_l, see Fig.1(e). If all duplicated genes in the duplicated genome are kept, i.e., no duplicated gene is lost (or $G_l = 0$), the genome duplication is called perfect. Otherwise, the genome duplication is called imperfect [13].

Usually, genome duplications occur much less than gene duplications [4, 13]. Furthermore, preferential-attachment principle is applied while adding new edges to the gene network, i.e., a new duplicated vertex is preferentially joined to those already well connected vertices in the network. The network is scale-free [4].

3 Dynamics of Gene Networks

Biological observations show that any organism contains a gene regulatory network starting from a simple initial state with fewer genes and gene-gene interactions. During evolution, through a series of gene duplications, new genes and gene-gene interactions are added into the network one by one. Occasionally, a genome duplication occurs, the genes are doubled and meanwhile the gene-gene interactions increase rapidly. On the other hand, after the genome duplication, most duplicated genes and duplicated edges are removed. A new state of the gene network is formed. The gene network keeps growing from the new state, and so on. We are interested in the growth dynamics of gene networks. The gene-gene interactions of gene networks play an important role in the dynamics.

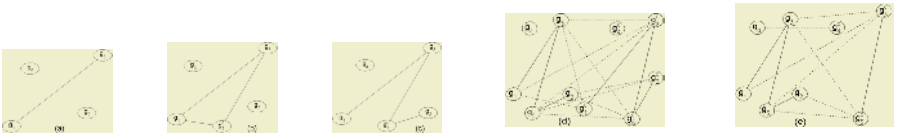


Fig. 1. Gene network growth: (a) A simple initial gene network with four genes, one pair of which has interaction relation. (b) Gene duplication: A gene is duplicated with a given rate and later lost with another rate. The new duplicated gene inherits all edges from its original gene together with an additional new edge connecting to the original gene. Here g_1 is duplicated into g_1^c. The edge $g_1^c g_3$ is inherent from the edge $g_1 g_3$. The new edge $g_1 g_1^c$ is created. (c) Edge adding and edge removing: Each duplicated gene undergoes a series of edge adding and edge removing based on two given rates, respectively. Here $g_1 g_1^c$ is removed and $g_1^c g_2$ is added. (d) Genome duplication: All vertices and edges are duplicated. (e) Gene loss in the duplicated genome with a given rate: The duplicated g_1^c and the duplicated g_2^c are removed from the duplicated genome

For any vertex v_i in a gene network, the degree of v_i refers to the number of its gene-gene interactions (i.e. edges). The degree distribution of v_i is defined as the probability $Pr(k)$ that v_i has k edges ($k \geq 0$). Denote d_i the number of the vertices that exactly have i edges. The network degree distribution is defined by all $d_i / \sum_{i=1}^{n} d_i$ ($0 \leq i \leq n$). Therefore, vertex degree distribution describes the local distribution of gene-gene interactions in networks, wheras network degree distribution describes the global distribution of gene-gene interactions in networks, i.e., how many genes exactly have i gene-gene interactions for all $0 \leq i \leq n$. It is shown in this section that the vertex degree distribution always follows power laws, but the network degree distribution may not (due to genome duplications), instead, it approaches to a power law tail.

3.1 Vertex Degree Distributions

Denote $n(t) = |N(t)|$ and $b(t) = |E(t)|$ the numbers of all vertices and edges of the network at time t, respectively. Suppose that at time t, the $i-$th vertex v_i has $x_i(t)$ edges in the network. While joining the new duplicated vertex to other vertices in the network, preferential-attachment principle is applied. It means that any new duplicated vertex is connected to v_i with the probability $\pi(x_i(t)) = \frac{x_i(t)}{\sum_{i=1}^{n(t)} x_i(t)}$. Because genome duplications can greatly change the degree distributions of vertices and networks, we have two cases.

Case 1 No genome duplication occurs

If no genome duplication occurs, vertices and edges are gradually added into the network by gene duplications. We have the following equations:

$$\begin{cases} \Delta n(t) &= (g_d - g_l)\Delta t, \\ \Delta b(t) &= (\lambda - \mu)\Delta n(t) = (\lambda - \mu)(g_d - g_l)\Delta t, \\ \Delta x_i(t) &= [\lambda \pi(x_i(t)) + \frac{\lambda}{n(t)} - \frac{2\mu}{n(t)}]\Delta t, \\ b(t) &= \frac{1}{2}\sum_{i=1}^{n(t)} x_i(t). \end{cases}$$

Similar to some previous studies [4, 11], we obtain that x_i follows a power law: There exist some constants c, k_0, and γ such that $Pr(x_i = k) = \frac{c}{(k+k_0)^\gamma}$, $k \geq 0$.

Since no genome duplication occurs, we similarly obtain that the network degree distribution also follows the same power law.

Case 2 One genome duplication occurs

If a genome duplication occurs, the network size is at first doubled and then decreased to a certain scale by the follow-up gene loss in the duplicated genome. Both the vertices and edges may be greatly changed. The original power law may also be invalid. For each i, suppose v_i^d is duplicated from v_i in the network. Assume v_i^d has $x_i^d(t)$ edges. Just after a genome duplication, $x_i^d = x_i$, both degrees increase from x_i to $2x_i + 1$.

(2a) A perfect genome duplication

If a perfect genome duplication occurs, the total vertices are exactly doubled. And the edges are also doubled together with one more new edge for each pair of the original genes and duplicated ones. Therefore, the vertex degree distribution keeps unchanged. We show this fact as follows:

$$\begin{cases} \Delta x_i(t) = \frac{\lambda(2x_i(t)+1)}{[4(\lambda-\mu)+1](g_d-g_l)t} + \frac{\lambda-2\mu}{2(g_d-g_l)t}, \\ \Delta x_i^d(t) = \frac{\lambda(2x_i^d(t)+1)}{[4(\lambda-\mu)+1](g_d-g_l)t} + \frac{\lambda-2\mu}{2(g_d-g_l)t}. \end{cases}$$

Similar to Case 1, the solution also has the form: $Pr(x_i = k) = Pr(x_i^d = k) = \frac{c}{(k+k_0)^\gamma}$,$k \geq 0$. It is still a power law distribution. This implies that after a perfect genome duplication, the vertex degree distribution still follows a power law, and so does the network degree distribution.

(2b) An imperfect genome duplication

If an imperfect genome duplication occurs, the duplicated genes in the duplicated genome are removed with a certain rate after the genome duplication. The network is taken as a new initial state to continue to grow.

Since a gene is lost with the rate G_l, it is kept in the network with the rate $G_k = 1 - G_l$. For each v_i, its degree number increases from x_i to $(1 + G_k)x_i$. For each v_i^d, if v_i^d is removed, then $x_i^d = 0$. If v_i^d is kept, $x_i^d = (1 + G_l)x_i$. We have

$$\begin{cases} \Delta x_i(t) = \frac{\lambda(1+G_k)x_i(t)}{2(1+G_k)(\lambda-\mu)(g_d-g_l)t} + \frac{\lambda-2\mu}{(1+G_k)(g_d-g_l)t}, \\ \Delta x_i^d(t) = \frac{\lambda(1+G_k)x_i^d(t)}{2(1+G_k)(\lambda-\mu)(g_d-g_l)t} + \frac{\lambda-2\mu}{(1+G_k)(g_d-g_l)t}. \end{cases}$$

It follows that its solution also has the form: $Pr(x_i = k) = Pr(x_i^d = k) = \frac{c}{(k+k_0)^\gamma}$, $k \geq 0$. Thus, the vertex degree distribution follows a power law.

3.2 Network Degree Distributions

After a genome duplication, some vertices may have doubled their degrees, but some may have decreased theirs. The gene-gene interactions in the network are greatly changed. The network degree distribution is then changed. Therefore, the power law of the network degree distribution may not hold. However, because the vertex degree distribution for all vertices still follows the power law, after a

long evolutionary time, the network degree distribution will somehow gradually recover to its power law state, which is shown as follows.

Suppose each vertex v_i has y_i edges ($i \geq 1$). From the above results, we can show that the vertex degree distribution follows the power law $p(x|y_i) = \frac{c}{x^\gamma}$ ($x \geq y_i$). Denote $p(x|y_i) = 0$ for all $x < y_i$. An initial network degree distribution $P_0(x)$ is defined by all $y_i / \sum_j y_j$. Let $P_i(x)$ denote the network degree distribution at the $i-$th gene duplication after the genome duplication. From the initial distribution $P_0(x)$, we recursively have $P_i(x) = \sum_{y=1}^{+\infty} p(x|y) P_{i-1}(y) (i \geq 1)$.

We show that $P_i(x)$ approaches to $p(x)$: $\lim_{i \to +\infty} P_i(x) = p(x)$ for $x \geq \bar{x}$, where $\bar{x} = \sum_{k \geq 1} kp(k)$. By induction, it follows that $P_i(x)$ has the form: $P_i(x) = \frac{c_i}{x^\gamma}$ for $x \geq 1$ (c_i may depend on x). We now show that $P_i(x) \to \frac{c}{x^\gamma}$ and the power law starts at the average vertex degree $\bar{x}$ of the gene network.

Table 1. $P_i(x)$ is cumulated by a series of power laws: $p(x|j)P_{i-1}(j)$

x	$p(x\|1)p_{i-1}(1)$	$p(x\|2)p_{i-1}(2)$	$p(x\|3)p_{i-1}(3)$	...	$p(x\|j)p_{i-1}(j)$	...
1	$\frac{c}{1^\gamma}\ \frac{c_{i-1}}{1^\gamma}$	-	-	-	-	...
2	$\frac{c}{2^\gamma}\ \frac{c_{i-1}}{1^\gamma}$	$\frac{c}{2^\gamma}\ \frac{c_{i-1}}{2^\gamma}$	-	-	-	...
3	$\frac{c}{3^\gamma}\ \frac{c_{i-1}}{1^\gamma}$	$\frac{c}{3^\gamma}\ \frac{c_{i-1}}{2^\gamma}$	$\frac{c}{3^\gamma}\ \frac{c_{i-1}}{3^\gamma}$	-	-	...
	... ...					
j	$\frac{c}{j^\gamma}\ \frac{c_{i-1}}{1^\gamma}$	$\frac{c}{j^\gamma}\ \frac{c_{i-1}}{2^\gamma}$	$\frac{c}{j^\gamma}\ \frac{c_{i-1}}{3^\gamma}$	-	$\frac{c}{j^\gamma}\ \frac{c_{i-1}}{j^\gamma}$	...
	... ...					

At first, each $P_i(x)$ is cumulated by a series of power law terms $p(x|j)P_{i-1}(j)$, i.e., $P_i(x) = \sum_{j=1}^{\infty} p(x|j)P_{i-1}(j)$ (see Table 1). Rewrite $P_i(x) = \frac{c}{x^\gamma} \sum_{j=1}^{x} \frac{c_{i-1}}{j^\gamma}$ (see Table 2). By induction principle, $\sum_{j=1}^{x} \frac{c_{i-1}}{j^\gamma} \to 1$. It follows that $P_i(x) \to \frac{c}{x^\gamma}$ as $i \to +\infty$, i.e. the network degree distribution approaches the power law.

Table 2. The coefficients of all terms for $P_i(x)$

Terms	Coefficients
$\frac{c}{1^\gamma}$	$\frac{c_{i-1}}{1^\gamma}$
$\frac{c}{2^\gamma}$	$\frac{c_{i-1}}{1^\gamma} + \frac{c_{i-1}}{2^\gamma}$
$\frac{c}{3^\gamma}$	$\frac{c_{i-1}}{1^\gamma} + \frac{c_{i-1}}{2^\gamma} + \frac{c_{i-1}}{3^\gamma}$
$\frac{c}{4^\gamma}$	$\frac{c_{i-1}}{1^\gamma} + \frac{c_{i-1}}{2^\gamma} + \frac{c_{i-1}}{3^\gamma} + \frac{c_{i-1}}{4^\gamma}$
...	... ...
$\frac{c}{j^\gamma}$	$\frac{c_{i-1}}{1^\gamma} + \frac{c_{i-1}}{2^\gamma} + \frac{c_{i-1}}{3^\gamma} + \frac{c_{i-1}}{4^\gamma} + \cdots + \frac{c_{i-1}}{j^\gamma}$
...	... ...

Next, we find the starting point of the power law of $\lim_{i \to +\infty} P_i(x)$. Each power law term $p(x|j)P_{i-1}(j)$ starts at j and its rate (i.e., probability) is $P_0(j)$, which comes from the initial network degree distribution defined by all y_i. Therefore, the expected starting point of the power law of $\lim_{i \to +\infty} P_i(x)$ is given by the expectation $\sum_{j=1}^{\infty} j\ P_0(j) = \bar{x}$, which is the average vertex degree of the gene network when the genome duplication is completed.

Therefore, after a genome duplication, the network degree distribution may not follow a power law. However, it gradually approaches to the power law of the vertex degree distribution over the tail $[\bar{x}, +\infty)$. The dynamics is as follows.

Network Dynamics *Let* $G = (V, E; g_d, g_l; \lambda, \mu; G_d, G_l)$ *be a gene network. Then*

(1) The vertex degree distribution always follows power laws. Genome duplications may change the power law exponents (See Fig.2 (a)(b)).

(1a) Before a genome duplication, it follows a power law: $p_1(k) = \frac{c_1}{(k+k_{01})^{\gamma_1}}$.

(1b) After a genome duplication, it may follow a new power law: $p_2(k) = \frac{c_2}{(k+k_{02})^{\gamma_2}}$.

(2) The network degree distribution has a power-law-tail property (See Fig.2(b)):

(2a) Before a genome duplication, it follows the power law of the vertex degree distribution: $p_1(k) = \frac{c_1}{(k+k_{01})^{\gamma_1}}$.

(2b) After a genome duplication, $P_i(k)$ *denotes the network degree distribution at the* $i-th$ *gene duplication. Then* $\lim_{i\to+\infty} P_i(k) = p_2(k),\ k \geq \bar{x} = \sum_{j\geq 1} jP_0(j)$.

It means that a genome duplication drives the network degree distribution away from its power law states. However, after an efficient large number of gene duplications, the network degree distribution gradually approaches to the power law of the vertex degree distribution over the tail starting at the average vertex degree when the genome duplication is completed, i.e., the tail $[\bar{x}, +\infty)$ *of the power law (see Fig.2(b)).*

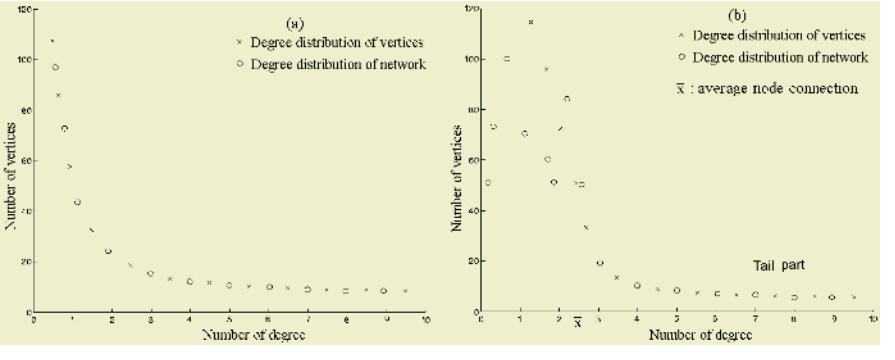


Fig. 2. Network growth: (a) Before a genome duplication: the degree distributions of vertices and network follow a power law. (b) After a genome duplication: the degree distribution of vertices follow a new power law, but the network degree distribution may not. It approaches to the power law of the vertex degree distribution over $[\bar{x}, +\infty)$

The dynamics shows the evolutionary process of a gene network: The network starts at a simple initial state and is enlarged by a series of gene duplications that occur during the network growing. Before a genome duplication, the degree distributions for both the vertices and the network follow the same power law. After a genome duplication, the vertex degree distribution follows a new power law, but the network degree distribution may not. However, after a large number of gene duplications, the network degree distribution gradually approaches to the

power law of the vertex degree distribution over the tail part, which starts at the average vertex degree. The gene network always repeats this growing process. A genome duplication may change the power law to a new one for the vertex degree distribution and drive the network degree distribution away from the power law states. The network always gradually recovers to the power law states.

4 Simulations

It is shown in last section that the (local) vertex degree distribution always follows power laws and the (global) network degree distribution may not, but approach a power law tail. In this section, we examine the dynamics of gene networks by computer simulations. A simulation program is designed for the purpose and is available at *http://xgu.zool.iastate.edu*.

The simulations are conducted for the model with the following parameters: $N_0 = 4,\ E_0 = 1,\ g_d = 1/n,\ g_l = 0,\ \lambda = 0.3,\ 0 \leq \mu \leq 0.1,\ G_l = 0.9$. One genome duplication occurs at $N = 2000$, i.e., $G_d \approx \frac{1}{8800} \approx 0.00011$. We focus on observing the vertex degree distributions and the network degree distributions through a series of simulations.

Initially, we start from a simple network consisting of four vertices with only one pair of them joined by an edge. Next, the network grows for 2000 gene duplications (one by one), followed by a genome duplication, which doubles the network size. Then, the network keeps only 10% of the genes in the duplicated genome and continue to grow until the network contains 9000 genes (i.e., about 8800 gene duplications occurs).

We take the average values for all parameters from ten groups of our simulations. By linear regressions on the log-log plots of the generated distributions from the simulations, we find that power law distributions best fit the data. Table 3 and Figure 3 show the vertex degree distributions and the network degree distributions. Table 3 also shows the average degrees and diameters of the networks, which are consistent with general biological systems [14–17].

Table 3 and Figure 3 show that: The vertex degree distribution always follows power laws with different exponents, but the network distribution may not. Under 2000 gene duplications, i.e., before the genome duplication, the network

Table 3. The parameters of the simulated gene network

Number of vertices N	Exponent for vertex degree distribution	Exponent for network degree distribution	Network average degree	Network average diameter
500	2.31	2.31	23.26	2.80
1000	2.16	2.16	28.91	3.12
2000	2.41	2.41	146.25	2.63
4000	2.48	N/A	198.58	2.50
5000	2.46	2.00*	219.38	2.51
9000	2.55	2.43*	401.21	2.50

$*$ This is the approximated power-law exponent of the tail $[\bar{x}, \infty)$.

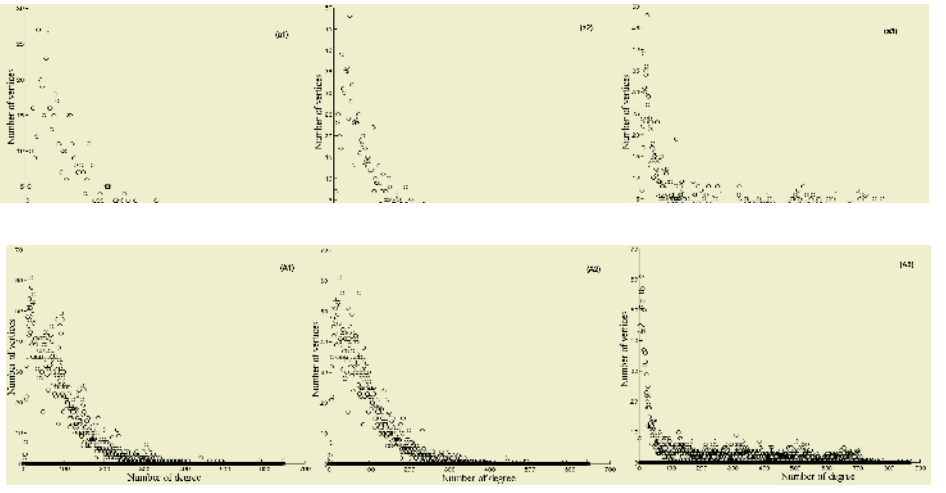


Fig. 3. Network growth process: (a1), (a2), and (a3) are the network degree distributions before genome duplication for N=500, 1000, and 2000, respectively. They follow power laws. (A1), (A2), and (A3) are the network degree distributions after a genome duplication for N=4000, 5000, and 9000, respectively. They may not be power laws, but very close to power law tails

degree distribution also follows power laws. However, when a genome duplication occurs at $n = 2000$, the network degree distribution does not follow a power law. The network keeps growing and the network degree distribution gradually recovers to its power law state. When the network contains 9000 genes, several thousands of gene duplications have occurred. The network degree distribution becomes very close to the power law of the vertex degree distribution over the tail starting at the average vertex degree. The network dynamics is verified.

5 Comparison and Discussion

Our gene network model has two advantages. At first, it includes more biological mechanisms [4, 13] (e.g., allowing genome duplications) and becomes more general and realistic than other previous ones [4, 13]. Secondly, our model and dynamics describe the dynamic behavior of gene networks. However, other previous ones contain only static behavior. Our dynamics shows that during the gene network growth, the vertex degree distribution always follows power laws with different exponents, but the network degree distribution may not. A genome duplication brings a new power law for the vertex degree distribution and drives the network degree distribution away from the power law state. However, after a sufficient long evolutionary time, the network degree distribution gradually recover to its power law state over the tail part. The gene networks under our mechanisms have a power-law behavior similar to those in [4, 15, 19]. After a genome duplication, the gene network continues to grow from a new state and

follows a new power law. The results again confirm that mechanisms decide dynamics, i.e., a power law exponent only depends on the network growth process, not the initial state of the network [4, 13]. In addition, our model and simulation provide an efficient way to explore large scale complex biological networks, whose large scale interactions are impossible to be dealt with by means of experiment. Moreover, the model can be generalized to deal with multiple organisms.

Acknowledgement

Research is supported by NIH Grant RO1 GM62118 (X.G.) and the research grant from the Plant Science Institute at Iowa State Univerity.

References

1. Kitano, H.: Systems Biology: A Brief Overview, *Science* 295(2002)1662-1664.
2. Strogatz, S. H.: Exploring complex networks, *Nature*, 410(2001)268-276.
3. Oltvai,Z., Barabási,A.-L.: Life's complexity pyramid, *Science*,298(2002)763-764.
4. Albert, B. and Barabási, A.-L.: Statistical mechanics of complex networks, *Review of Modern Physics*, 74(2002)47-97.
5. Sali, S.: Functional links between proteins, *Nature*, 402(1999)23-26.
6. Wagner, A.: The yeast protein interaction network evolves rapidly and contains few redundant duplicate genes, *Molecular Biology and Evolution*, 18(2001)1283-1292.
7. Wagner, A.: How large protein interaction networks evolve. *Proc. R. Soc. Lond.*, B 270(2003)457-466.
8. Uetz, P., Giot, L., etc.: A comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae, *Nature*, 403(2001)623-627.
9. Jeong, H., Tombor, B., Albert, R., Oltvai, Z.N., and Barabási, A.-L.: The large-scale organization of metabolic networks,*Nature* 407(2000)651-654.
10. Ravasz, E., Somera, A., Mongru, D. A., Oltvai, Z. and Barabási, A.-L.: Hierarchical organization of modularity in metabolic networks, *Science*, 297(2002)1551-1555.
11. Barabási, A. and Albert, A.-L.: Emergence of scaling in random networks, *Science*, 286(1999)509-512.
12. Azevedo, R. and Leroi, A. M.: A power law for cells, *PNAS*, 98(2001)5699-5704.
13. Chung, F., Lu, L., Dewey, T. G., and Galas, D. J.: Duplication Models for Biological Networks, *Journal of Computational Biology*, 10(2002)677-687.
14. Albert, R., Jeong, H. and Barabási, A.-L.: Diameter of the World Wide Web, *Nature*, 401(1999)130-131.
15. Chung, F. and Lu, L.: The average distances in random graphs with given expected degrees, *PNAS*, 99(2002)15879-15882.
16. Givan, M. and Newman, E. J.: Community structure in social and biological networks, *PNAS*, 99(2002)7821-7826.
17. Watts, D. J. and Strogatz, H.: Collective dynamics of 'small-world' networks, *Nature*, 393(1998)440-442.
18. Ohno, S.: *Evolution by Gene Duplication*, Springer, New York (1970).
19. Bhan, A., Galas, D. J. and Dewery, T. G.: A duplication growth model of gene expression network, *Bioinformatics*, 18(2002)1486-1493.
20. Wolfe, K. and Shields, D.: Molecular evidence for an ancient duplication of the entire yeast genome, *Nature*, 387(1997)708-13.
21. Stillman, B.: Genomic Views of Genome Duplication, *Science*, 294(2001)2301-2304.

Conserved Interval Distance Computation Between Non-trivial Genomes*

Guillaume Blin[1] and Romeo Rizzi[2]

[1] LINA FRE CNRS 2729 Université de Nantes, 2 rue de la Houssinière
BP 92208, 44322 Nantes Cedex 3, France
blin@lina.univ-nantes.fr
[2] Università degli Studi di Trento, Dipartimento di Informatica e Telecomunicazioni
Via Sommarive, 14, I38050 Povo, Trento (TN), Italy
Romeo.Rizzi@unitn.it

Abstract. Recently, several studies taking into account the ability for a gene to be absent or to have some copies in genomes have been proposed, as the examplar distance [6, 11] or the gene matching computation between two genomes [3, 10]. In this paper, we study the time complexity of the conserved interval distance computation considering duplicated genes using both those two strategies.

Keywords: Conserved interval distance, Exemplar string, Matching, Computational complexity, Longest Common Substring, Duplicated genes.

1 Introduction

In comparative genomics, gene order study in a set of organisms has been intensively led essentially in phylogenetic research field [2, 4, 5]. Most of the methods associated to gene order study are based on a distance computation. This distance has to reflect the number of genetic operations needed to transform a source genome into a target genome. For this purpose, a set of distances and associated methods have been developed in the past decade. Among others, we can mention three intensively studied distances: *edit* [9, 12], *breakpoint* [3], and *conserved interval* [1] distances.

From an algorithmic point, distances can roughly be defined as follows: given a set $\mathcal{F}$ of *gene families*, two *genomes* G and H, represented as sequences of signed elements (genes) from $\mathcal{F}$, and a set of evolutionary operations that operate on segments of genes (like reversals, transpositions, insertions, duplications, deletions for example), the distance between G and H is the minimum number of operations needed to transform G into H.

Until recently, the assumption that *in a genome there is no copy of a gene* was a requirement of most of the methods associated to gene order study. This restriction reduces the problem to the comparison of signed permutations [8]. It

* This work was partially supported by the French-Italian PAI Galileo project number 08484VH

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 22–31, 2005.

is known that this assumption is very restrictive and is only justified in small virus genomes, therefore one needs to consider genomes containing duplicated genes.

In [11], Sankoff has proposed a method to select, from the set of copies of a gene, the common ancestor gene such that the distance between the reduced genomes is minimized. In [6], Bryant proved that the corresponding problem, so called *exemplar string*, was **NP**-complete for two distances: the signed reversals and the breakpoint distances. Marron *et al.* have proposed in [12] methods relying on a matching between genes of two genomes. Provided with a matching between genes of the two genomes, one can, by a rewriting of the genomes according to the matching, create genomes without duplicated genes and solve the reduced problem.

In this paper, we investigate the complexity of both the use of exemplar strategy and of matchings to compute the conserved interval distance between genomes containing duplicated genes. First we prove that the use of both strategies unfortunately induces **NP**-completeness. To overstep **NP**-hardness of problems, many techniques have been developed: heuristic, parameterized complexity and approximation algorithm. For biological problems those alternative techniques have been intensively used, since in most cases specific properties of the problem are not taken into account in the **NP**-hardness proof.

This paper is organized as follows. After presenting some preliminaries in Section 2, we show in Section 3 that both the use of exemplar strategy and of matchings to compute the conserved interval distance between genomes containing duplicated genes induces **NP**-completeness. Then in Section 4, we present a heuristic approach based on the Longest Common Substring which have been implemented and tested over a set of 20 bacteria.

2 Preliminaries

Genomes, Gene Families and Gene. Following terminology introduced in [11], a *genome* G is a sequence of elements of an alphabet $\mathcal{F}$ (referred as the set of *gene families*) such that each element is provided with a sign ($+$ or $-$). Each occurrence of a gene family from $\mathcal{F}$ in G is called a *gene*. Given a genome $G = g_1 g_2 \dots g_n$, we say that gene g_i precedes gene g_{i+1}. For two genomes G and H and a gene family $\mathbf{f}$, the number of occurrences of $\mathbf{f}$ in G and H is called the *cardinality* of family $\mathbf{f}$. A gene family $\mathbf{f}$ is said to be *trivial* if $\mathbf{f}$ has cardinality exactly 1 or 2. Otherwise, $\mathbf{f}$ is said to be *non-trivial*. A gene belonging to a trivial (resp. non-trivial) family is said to be trivial (resp. non-trivial). A segment (*i.e.* a substring) of G that contains only non-trivial genes is called a *non-trivial segment*. We say that two genomes G and H are *balanced* if, for any gene family $\mathbf{f}$, there are as many occurrences of $\mathbf{f}$ in G as in H.

Conserved Interval, Conserved Interval Distance. Following terminology introduced in [1], given a set of n genomes $\mathcal{G}$ and two genes $a, b \in \mathcal{F}$, an interval $[a, b]$ is a *conserved interval* of $\mathcal{G}$ if (1) either a precedes b, or -b precedes -a in each genome of $\mathcal{G}$ and (2) the set of unsigned genes (*i.e.* not considering signs)

appearing between genes a and b is the same for all genomes of $\mathcal{G}$. For example, given two genomes $G_1 = a\ b\ c\ g\ e\ f\ \text{-}d\ h$ and $G_2 = a\ g\ \text{-}c\ \text{-}b\ e\ \text{-}f\ \text{-}d\ h$, there are seven conserved intervals between G_1 and G_2: $[a, \text{-}d]$, $[a, e]$, $[a, h]$, $[b, c]$, $[e, \text{-}d]$, $[e, h]$ and $[\text{-}d, h]$.

Given two set of genomes $\mathcal{G}$ and $\mathcal{H}$, the *conserved interval distance* between $\mathcal{G}$ and $\mathcal{H}$ is defined by $d(\mathcal{G}, \mathcal{H}) = N_{\mathcal{G}} + N_{\mathcal{H}} - 2N_{\mathcal{G} \bigcup \mathcal{H}}$ where $N_{\mathcal{G}}$ (resp. $N_{\mathcal{H}}$ and $N_{\mathcal{G} \bigcup \mathcal{H}}$) is the number of conserved intervals in $\mathcal{G}$ (resp. $\mathcal{H}$ and $\mathcal{G} \bigcup \mathcal{H}$). For example, let $\mathcal{G} = \{G_1, G_2\}$ and $\mathcal{H} = \{H_1, H_2\}$ be two sets of genomes where G_1 and G_2 are as above and $H_1 = a\ e\ \text{-}f\ b\ g\ c\ \text{-}d\ h$ and $H_2 = a\ f\ \text{-}c\ \text{-}g\ b\ \text{-}e\ \text{-}d\ h$. We obtain $d(\mathcal{G}, \mathcal{H}) = 7 + 3 - 4 = 6$. In the rest of the paper, for readability, we denote the conserved interval distance between two singleton sets $d(\{G\}, \{H\})$ by $d(G, H)$.

Gene Matching. Let $G = g_1 g_2 \dots g_n$ and $H = h_1 h_2 \dots h_m$ be two genomes on $\mathcal{F}$. A *gene matching* $\mathcal{M}$ between G and H is a maximal matching between genes of G and H such that, for every pair $(g_i, h_j) \in \mathcal{M}$, g_i and h_j belong to the same family. By maximal matching, we mean that for any gene family $\mathbf{f}$, it is forbidden to have at the same time an occurrence of $\mathbf{f}$ in G and one in H that do not belong to $\mathcal{M}$. It follows from the maximality condition of matchings that in any matching $\mathcal{M}$ between balanced genomes G and H, every gene of G is matched to a gene of H and conversely. Given a matching $\mathcal{M}$ and a segment $s = s_1 s_2 \dots s_m$ of G if, for all $1 \leq i \leq m$, $(s_i, t_i) \in \mathcal{M}$ such that: (1) $s_i = t_i$ and $t = t_1 t_2 \dots t_m$ is a segment of H or (2) $s_i = -t_i$ and $t = t_m t_{m-1} \dots t_1$ is a segment of H then s is *perfectly matched* in $\mathcal{M}$; *not-perfectly matched* otherwise.

Minimum Conserved Interval Matching. Given two genomes G, H and a gene matching $\mathcal{M}$, we denote by $d(G, H, \mathcal{M})$ the conserved interval distance between G and H with respect to $\mathcal{M}$, and by $d(G, H)$ the conserved interval distance between G and H, defined as the minimum $d(G, H, \mathcal{M})$ among all matchings $\mathcal{M}$. The matching $\mathcal{M}$ such that $d(G, H, \mathcal{M}) = d(G, H)$ is a *minimum conserved interval matching.*

3 Hardness Results

In this section, we first prove that, even if there is no non-trivial segment containing more than one gene, Exemplar Conserved Interval Distance (ECID) problem (formalized hereafter) is **NP**-complete. Then, we prove that, even with just one non-trivial gene family, the problem of finding a Minimum Conserved Interval Matching (MCIM) is **NP**-complete. This last result is based on a polynomial time reduction which is inspired from the one presented in [3] which proves that a connected problem, Minimum Breakpoint Matching, is **NP**-complete.

Theorem 1. Exemplar Conserved Interval Distance *problem is* **NP***-complete even when all non trivial segments are composed of only one duplicated gene.*

To prove the correctness of Theorem 1, we provide a polynomial time reduction from the **NP**-complete problem Minimum Set Cover [7]. Following terminology introduced in [7], we recall that given a collection C of subsets of a finite set E, a *cover* for E is a subset $C' \subseteq C$ such that every element of E belongs to at least one member of C'. For the sake of clarity, we now state formally the two decision problems we consider: ECID and Minimum Set Cover. Given two genomes G and H, and a positive integer k, ECID problem asks whether it is possible to find two exemplar genomes G' and H' of resp. G and H such that $d(G', H') \leq k$. Given a collection C of subsets of a finite set E and a positive integer k', Minimum Set Cover problem asks whether C contains a cover C' for E s.t. $|C'| \leq k'$.

Hereafter, we consider, w.l.o.g., that in each $C_i \in C$, any $e_j \in C_i$ is also in C_j with $1 \leq i, j \leq m$ and $i \neq j$. In fact, by definition, if an element is covered by only one subset then this subset must be part of C'. In the following, we will prove that, even if G does not contain more than one occurrence of each duplicated gene, ECID problem is **NP**-complete. Given an integer k', two exemplar genomes G' and H', one can compute polynomially $d(G', H')$ and check if $d(G', H') \leq k'$ (see [1]). Therefore, ECID problem is in **NP**. The remainder of the section is devoted to proving that it is also **NP**-hard. For this purpose, we reduce Minimum Set Cover problem to ECID problem. Let $C = \{C_1, C_2 \dots C_m\}$ be a collection of m subsets of a finite set $E = \{e_1, e_2 \dots e_n\}$ of n elements.

In the rest of this section, we consider that $\mathcal{F} \subseteq \mathbb{N}$ but any genome is built with elements of $\mathcal{F}$ provided with signs (*i.e.* $\mathbb{R}$). In other words, genes 3 and -3 are of the same family. Let us detail the construction of the two genomes G and H. Let $y = |E| + 2$ if $|E|$ is even, $y = |E| + 1$ otherwise. Let $z_i = (y + 2).(i - 1)$ for any $1 \leq i \leq m + 1$. From (C, E), we construct two genomes G and H as described below (an illustration is given in Figure 1):

$$G_1 = \gamma_{|E|+1}\ \gamma_{|E|+2} \cdots \gamma_{|E|+m-1} \alpha_1\ \beta_1\ \dots \alpha_m\ \beta_m\ \gamma_1\ \gamma_{|E|+m}\ \gamma_2\ \gamma_{|E|+m+1} \cdots \gamma_{2|E|+m-1}\ \gamma_{|E|}$$

$$H_1 = \alpha_1\ \theta_1\ \gamma_{|E|+1}\ \alpha_2\ \theta_2\ \gamma_{|E|+2} \cdots \gamma_{|E|+m-1}\ \alpha_m\ \theta_m\ \gamma_{|E|+m}\ \gamma_{|E|+m+1}\ \cdots\ \gamma_{2|E|+m-1}$$

We now detail the substrings that compose G_1 and H_1:

- for $1 \leq i \leq m$, we construct the sequences of genes $\alpha_i = z_i$ and $\beta_i = z_i{+}1\ z_i{+}2 \dots z_i{+}y{+}1$;
- for $1 \leq i \leq 2|E| + m - 1$, we construct a gene $\gamma_i = z_{m+1} + i$;
- for $1 \leq i \leq m$, we construct a gene $\theta_i = z_i{+}2\ z_i{+}4 \dots z_i{+}y\ z_i{+}1\ z_i{+}3 \dots z_i{+}y{-}1\ z_i{+}y{+}1$.

Note that G_1 and H_1 are two exemplar genomes. Genome G is, thus, a copy of G_1. We now turn to transform H_1 into a non-exemplar genome H: for $1 \leq i \leq m$ and $1 \leq j \leq |E|$, if $e_j \in C_i$ then gene γ_j is inserted between the j^{th} and the $j{+}1^{th}$ genes of θ_i. We denote by ECID-construction any construction of this type. An illustration of an ECID-construction is given in Figure 1. Intuitively, θ_i is a shuffle of β_i with some inserted γ_js (*i.e.* no conserved adjacencies) for $1 \leq i \leq m$. Clearly, our construction can be carried out in polynomial time. Moreover, the result of such a construction is indeed an instance of ECID problem.

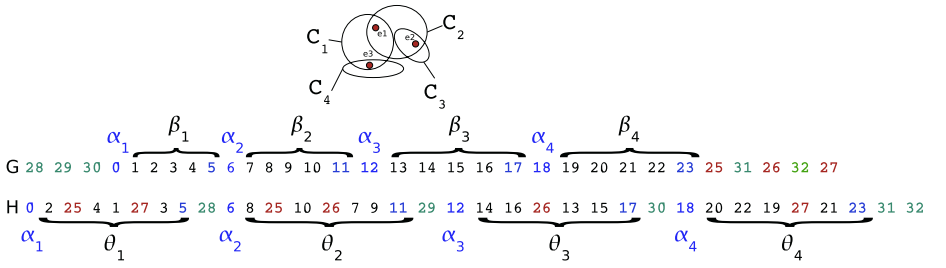


Fig. 1. Example of an ECID-construction where $E = \{e_1, e_2, e_3\}$ and $y = 4$

We now turn to proving that our construction is a polynomial time reduction from Minimum Set Cover to ECID problem where G is an exemplar genome whereas H is not. Let first note that, by construction, there are only $|E|$ duplicated gene families in G and H, namely the γ_is for $1 \leq i \leq |E|$.

Lemma 1. *The only conserved intervals that can exist between G and any exemplar genome H' of H are intervals $[\alpha_i, z_i + y + 1]$ such that all the γ_js of $[\alpha_i, z_i + y + 1]$ in H' have been deleted, with $1 \leq i \leq m$ and $1 \leq j \leq |E|$.*

Lemma 2. *Let $I = (C, E, k')$ be an instance of the problem* Minimum Set Cover *with a collection $C = \{C_1, C_2 \dots C_m\}$ of m subsets of a finite set $E = \{e_1, e_2 \dots e_n\}$, and $I' = (G, H, k)$ an instance of* ECID *problem obtained by an* ECID*-construction from I. C contains a cover C' of E of size less than or equal to k' iff $d(G, H') \leq k$ where H' is an exemplar genome of H and $k = |G|.|G-1| - 2(m - k')$.*

Proof. ($\Rightarrow$) Suppose C contains a cover C' of E of size less than or equal to k'. Let $f : e_i \rightarrow \{C_1, C_2, \dots C_m\}$ be the function which, given an element of E, returns the index of the subset covering this element in C'. Let $I' = (G, H, k)$ be the instance obtained from an ECID-construction of I. We look for an exemplar genome H' of H such that $d(G, H') \leq k$. We define H' as follows: for each $e_j \in E$, delete γ_j of θ_p for all $p \in \{1, 2, .., m\}/\{f(e_j)\}$.

By construction, E denotes the set of duplicated gene families and by construction the only duplicated genes in H are the γ_is. Therefore, H' is exemplar since one deletes all occurrences but one of γ_i with $1 \leq i \leq |E|$. Remains us to prove that $d(G, H') \leq k$. By definition, for each $C_j \notin C'$ and each $e_i \in C_j$, $f(e_i) = p$ with $p \neq j$ and γ_i of θ_j has been deleted. Since all the γ_is of θ_j in H' have been deleted, there is a conserved interval $[\alpha_j, z_j + y + 1]$ between G and H'. Globally, there are at least $m - k'$ such subsets. Therefore, there are at least $m - k'$ conserved intervals between G and H'. Thus, $d(G, H') \leq |G|.|G-1| - 2(m - k')$, since the number of conserved intervals between a genome G and itself is $\frac{|G|.|G-1|}{2}$ and $|G| = |H'|$.

($\Leftarrow$) Suppose we have an exemplar genome H' of H such that $d(G, H') \leq k$. Assume, w.l.o.g., that $d(G, H') = d' \leq k$. We define C' as follows: in H', if $\gamma_j \in \theta_p$ then $f(e_j) = p$ and $C_p \in C'$. We now turn to proving that C' defines

a cover of E of size at most k'. Since H' is an exemplar genome of H, there is exactly one occurrence of each gene family in H'. Therefore, C' contains a set of subsets that covers E. Remains us to prove that $|C'| \leq k'$.

By definition, since $d' \leq |G|.|G-1| - 2(m-k')$, there are at least $m-k'$ conserved intervals between G and H'. By Lemma 1, the only conserved intervals that can exist between G and any exemplar genome H' of H are intervals $[\alpha_i, z_i + y + 1]$ such that all the γ_js of $[\alpha_i, z_i + y + 1]$ in H' have been deleted. Therefore, by construction, there are at least $m-k'$ such intervals $[\alpha_i, z_i + y + 1]$ in H'. Correctness of Theorem 1 follows. □

Given Theorem 1, one can ask if instead of deleting the duplicated genes, one can compute the interval distance taking into account duplicated genes [3, 12]. For this purpose, we propose the MCIM problem: finding a minimum conserved interval matching between two genomes. Unfortunately, as we will show hereafter, this problem is also **NP**-complete.

Theorem 2. Minimum Conserved Interval Matching *problem is* **NP***-complete.*

To prove the correctness of Theorem 2, we provide a polynomial time reduction from the **NP**-complete problem Minimum Bin Packing [7]. For the sake of clarity, we now state formally the two decision problems we consider: MCIM and Minimum Bin Packing. Given two genomes G and H, and an integer k, MCIM problem asks whether it is possible to find a matching between G and H such that $d(G,H) \leq k$. Given a finite set $U = \{u_1, u_2, \ldots, u_n\}$, a size $s(u) \in \mathbb{Z}^+$ for each $u \in U$ and two positive integers k' and $\mathcal{C}$, Minimum Bin Packing problem asks whether there is a partition of U into k' disjoint sets $U_1, U_2, \ldots, U_{k'}$ such that $\sum(s(u)|u \in U_i) \leq \mathcal{C}$ for each U_i.

It is easily seen that MCIM is in **NP** since given an integer k and a set of matchings between two genomes we can polynomially compute the number of conserved intervals between G and H and thus check if the distance is less than or equal to k (see [1]). The remainder of the section is devoted to proving that MCIM is also **NP**-hard even when there is only one non trivial family in $\mathcal{F}$, which implies Theorem 2. For this, we reduce Minimum Bin Packing problem to MCIM problem. Let $\mathcal{N} = k'.\mathcal{C} - \sum_{i=1}^{n} s(u_i)$.

Let us first detail the construction of genomes G and H from a Minimum Bin Packing instance $(U, k', \mathcal{C})$. The gene families are $\mathcal{F} = \{\alpha, \beta, \mathbf{x}, A_1, A_2 \ldots, A_{n+\mathcal{N}}, B_1, B_2, \ldots, B_{k'+1}\}$. On the whole, there are $k' + \mathcal{N} + n + 4$ families of genes, and $\mathbf{x}$ is the unique non-trivial family. For $1 \leq i \leq n$ (resp. $n+1 \leq i \leq n+\mathcal{N}$), we denote by $\mathbf{u'_i}$ a sequence of $s(u_i)$ consecutive genes $\mathbf{x}$ (resp. one gene $\mathbf{x}$). For $1 \leq j \leq k'$, $\mathbf{U'_j}$ represents a sequence of $\mathcal{C}$ consecutive genes $\mathbf{x}$. G and H are defined as follows:

$G = \alpha\ \mathbf{u'_1}\ A_1\ \mathbf{u'_2}\ A_2\ \ldots\ \mathbf{u'_{n+\mathcal{N}}}\ A_{n+\mathcal{N}}\ B_1\ B_2\ \ldots\ B_{k'+1}\ \beta$

$H = \alpha\ B_1\ \mathbf{U'_1}\ B_2\ \mathbf{U'_2}\ \ldots\ B_{k'}\ \mathbf{U'_{k'}}\ B_{k'+1}\ A_1\ A_2\ \ldots\ A_{n+\mathcal{N}}\ \beta$

An illustration of such a construction, that can obviously be achieved in polynomial time, is given in Figure 2. To complete the construction of the instance

of MCIM, it remains to us to define k: $k = \frac{|G|.(|G|-1)}{2} + \frac{|H|.(|H|-1)}{2} - 2q$ with $q = 1 + \sum_{i=1}^{n} \frac{s(u_i).(s(u_i)-1)}{2}$

In the next three lemmas, that establish that MCIM is **NP**-complete, we consider an instance $(U, k', \mathcal{C})$ of Minimum Bin Packing and the corresponding instance (G, H, k) of MCIM according to the above construction.

Lemma 3. *Given a matching $\mathcal{M}$, a non trivial segment of size p perfectly matched in $\mathcal{M}$ induces more conserved intervals (*i.e. $\frac{p(p-1)}{2}$*) than a non-trivial segment of size k not-perfectly matched.*

Lemma 4. *$d(G, H) \geq k$ and in any matching $\mathcal{M}$ between G and H, any conserved interval I with respect to $\mathcal{M}$ is either $[\alpha, \beta]$ or $I = [p, q]$ with $S[p..q]$ being a non trivial segment.*

Lemma 5. *There is a partition of U into k' disjoint sets $U_1, U_2, \ldots, U_{k'}$ such that the sum of the sizes of the elements in each U_i is at most $\mathcal{C}$ if and only if $d(G, H) \leq k$.*

Proof. ($\Leftarrow$) Suppose that $d(G, H) \leq k$. By Lemma 4, we know that $d(G, H) = k$, and any conserved interval I with respect to a minimum conserved interval matching between G and H is either $[\alpha, \beta]$ or $I = [p, q]$ with $S[p..q]$ being a non trivial segment. Moreover, if $d(G, H) = k$ then the number of conserved intervals should be maximal (*i.e.* $1 + \sum_{i=1}^{n} \frac{s(u_i).(s(u_i)-1)}{2}$). Therefore, by Lemma 3, any non-trivial segment u'_i, with $1 \leq i \leq n$, in G should be matched with a sequence of consecutive genes $\mathbf{x}$ in H. Precisely, for any $1 \leq i \leq n$, there is a given $1 \leq j \leq k'$ such that the sequence u'_i is perfectly matched with a substring of U'_j as illustrated in Figure 2.

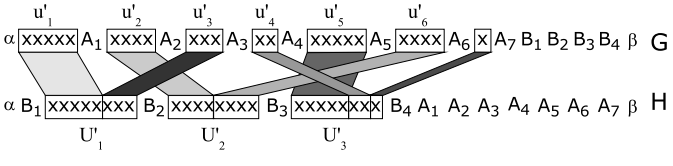


Fig. 2. Instance of MCIM associated to the Minimum Bin Packing instance where $k' = 3$, $\mathcal{C} = 8$ and $U = \{u_1, \ldots, u_6\}$ with $s(u_1) = s(u_5) = 5$, $s(u_2) = s(u_6) = 4$, $s(u_3) = 3$ and $s(u_4) = 2$ and the gene to gene matching corresponding to the following partition of U : $U_1 = \{u_1, u_3\}$, $U_2 = \{u_2, u_6\}$ and $U_3 = \{u_4, u_5\}$

Therefore, such a matching induces a partition P of the set of sequences $\{u'_1, u'_2, \ldots, u'_n\}$ into at most k' disjoint sequences $U'_1, U'_2, \ldots, U'_{k'}$. As, by construction, $|U'_i| = \mathcal{C}$ for $1 \leq i \leq k'$, P corresponds to an answer to the corresponding Minimum Bin Packing instance.

($\Rightarrow$) Suppose we have a partition P of U into disjoint sets $U_1, U_2, \ldots, U_{k'}$ each of cardinality at most $\mathcal{C}$. We compute a gene matching $\mathcal{M}$ between G and H as follows: (1) each trivial gene in G is matched with its corresponding gene

in H and (2) for $1 \leq j \leq k'$, for each $u_i \in U$, if $u_i \in U_j$, then the sequence of genes $\mathbf{x}$ of u'_i in G is perfectly matched with the first free (*i.e.* not already matched) sequence of genes $\mathbf{x}$ of U'_j in H.

Since $\mathcal{M}$ is built according to P, we claim that each non-trivial segment u'_i, with $1 \leq i \leq n$, is matched to a contiguous sequence of genes $\mathbf{x}$ in H. Thus, any non-trivial segment u'_i, with $1 \leq i \leq n$, in G induces $\frac{s(u_i).(s(u_i)-1)}{2}$ conserved intervals. Therefore, $\mathcal{M}$ induces $1 + \sum_{i=1}^{n} \frac{s(u_i).(s(u_i)-1)}{2}$ conserved intervals (see proof of Lemma 4). This leads to $d(G, H, \mathcal{M}) \leq k$, and so to $d(G, H) \leq k$. Correctness of Theorem 2 follows. □

4 Using the L.C.Substring to Approximate MCIM

In this section, we present a heuristic approach to solve MCIM problem that performs well on real data. This approach uses the following intuition: *long segments of genes that match, up to a complete reversal, in two genomes are likely to belong to a Minimum Conserved Interval Matching.* Remark that given two trivial genomes this intuition gives the optimal solution. Unfortunately, this is not always the case when considering non trivial genomes. Even so, this approach works very well on real genomes. In the rest of this section, we consider two genomes G and H build with elements of $\mathcal{F} \subseteq \mathbb{N}$ and provided with signs.

Our approach to solve MCIM problem is based on the following loop:

1. Identify a longest common segment s of genes between G and H (by common segment, we mean a segment appearing, up to a complete reversal, both in G and H);
2. Replace s in both genomes by an integer g_c, further call *compressed gene*, s.t. $g_c \notin \mathcal{F}$ (this induces that g_c is a trivial gene);
3. Mark g_c as treated;
4. Store the number of genes of s in $N_s[g_c]$ and the set of genes of s in $C[g_c]$.

While a common segment of unmarked genes exists, the algorithm performs the loop described above on the modified genomes. In the following, we will refer to the modified versions of the genomes G and H as G' and H'. Once the algorithm exits of the loop, any unmarked gene g_u of G' and H' is deleted since, by definition, g_u is not common (*i.e.* there are more genes g_u in one of the genomes). This first step of the algorithm leads to the computation of two exemplar genomes G' and H' of resp. G and H. Clearly, this step gives the gene to gene matching of G and H in a compressed version: the corresponding matching $\mathcal{M}$ is obtain by perfectly matching the segments of genes corresponding to each compressed gene of G as illustrated in Figure 3.

In a second step, the algorithm computes the interval distance induced by $\mathcal{M}$. By Lemma 3, each compressed gene g_c of G' induces $\frac{k(k-1)}{2}$ conserved intervals between G and H with $k = N_s[g_c]$. Moreover, some conserved intervals between G' and H' may exist. Therefore, since G' and H' are trivial genomes, the algorithm computes the set of conserved intervals S_{ci} between G' and H' in polynomial time using the algorithm defined in [1]. Since G' and H' are composed

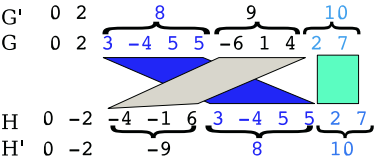


Fig. 3. The matching of the genes of G and H deduced from the exemplar genomes G' and H'

of compressed genes, for each conserved interval $[g_{c1}, g_{c2}] \in S_{ci}$, $N_s[g_{c1}].N_s[g_{c2}]$ conserved intervals between G and H are induced. Indeed, if $[g_{c1}, g_{c2}] \in S_{ci}$ then a segment of genes $g_{c1}\lambda g_{c2}$ appears in G' and either a segment of genes $g_{c1}\lambda' g_{c2}$ or $-g_{c2}\lambda' - g_{c1}$ appears in H' with λ and λ' being similar segments of genes not considering genes order and sign. Therefore, considering $\mathcal{M}$, for any genes $g_i \in C[g_{c1}]$ and $g_j \in C[g_{c2}]$, $[g_i, g_j]$ is a conserved interval between G and H. This step returns $d(G, H, \mathcal{M}) = |G|.|G-1| - 2(\sum_{g_c \in G'} \frac{N_s[g_c].(N_s[g_c]-1)}{2} + \sum_{[g_{c1},g_{c2}] \in S_{ci}} N_s[g_{c1}].N_s[g_{c2}])$

We implemented our approach using a suffix tree. Indeed, longest common segments between G and H can be found in linear time by browsing a suffix tree built on G, H and the reversed of H. To test our algorithm and get an estimate of its performance in practice, we applied our heuristic approach to a set of 20 bacteria from NCBI.

Data and programs used and mentioned in this article can be found at
`http://www.sciences.univ-nantes.fr/info/perso/permanents/blin/Cocoon05/`

Interesting characteristics of this set are given on the web page. We implemented the brute force algorithm which consists in computing all possible matchings and we compared the obtained results. In average, the gap between the optimal solution *opt* and the solution given by our algorithm is less than $0,12\%$ of *opt*. We noticed that more than $\frac{2}{3}$ of the bacteria have duplicated genes with, for most of them, duplicated families of cardinality 2. The effectiveness of our algorithm relies on the fact that the number of duplicated genes are not significant compared to the size of the genomes. Moreover, since our algorithm gives the optimal solution for trivial genomes, duplicated genes have a very little impact on the results.

5 Conclusion

The assumption of uniqueness of each gene in a genome has been a requirement of traditional methods in comparative genomics but is only justified in small virus genomes, since in general, there are more than one copy of a gene in a genome. In this paper, we investigate the time complexity of the conserved interval distance computation considering duplicated genes. We proved that both use of exemplar and matching methods leads to **NP**-completeness. We are doing thorough experimental testing which will determine how well our algorithm does in practice under different regimes of duplication, but our preliminary results are extremely encouraging.

Note that, since the Brute Force Algorithm is in $O(k^{k.l}n)$ with k being the maximal cardinality of any non-trivial gene family, l being the number of non-trivial families and n being the size of the genomes, MCIM problem is in **FPT** for parameter k and l. In order to be usable in many reconstruction algorithms, it would be of interest to determine if the problem is in **FPT** for other parameters.

References

1. A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. *Proceedings of COCOON 03*, 2697 of LNCS:68–79, 2003.
2. M. Blanchette, T. Kunisawa, and D. Sankoff. Gene order breakpoint evidence in animal mitochondrial phylogeny. *J. Mol. Evol.*, 49(2):193–203, 1999.
3. G. Blin, C. Chauve, and G. Fertin. The breakpoints distance for signed sequences. In *Actes de* CompBioNets 2004, volume 3 of *Texts in Algorithms*, pages 3–16. KCL Publications, 2004.
4. G. Bourque and P. A. Pevzner. Genome-scale evolution: Reconstructing gene orders in the ancestral species. *Genome Res.*, 12(1):26–36, 2002.
5. G. Bourque, P.A. Pevzner, and G. Tesler. Reconstructing the genomic architecture of ancestral mammals: lessons from human, mouse and rat genomes. *Genome Res.*, 14(4):507–516, 2004.
6. D. Bryant. The complexity of calculating exemplar distances. In D. Sankoff and J. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, pages 207–212. Kluwer Acad. Pub., 2000.
7. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
8. O. Gascuel, editor. *Mathematics of Evolution and Phylogeny*. Oxford Univ. Press, 2004. To appear.
9. M. Marron, K.M. Swenson, and B.M.E. Moret. Genomic distances under deletions and inversions. *Proceedings of COCOON 03*, 2697 of LNCS:537–547, 2003.
10. B. M. E. Moret, A. C. Siepel, J. Tang, and T. Liu. Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. In *WABI 2002*, volume 2452 volume of LNCS, pages 521–536. Springer Verlag, 2002.
11. D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.
12. K.M. Swenson, M. Marron, J.E Earnest-DeYoung, and B.M.E. Moret. Approximating the true evolutionary distance between two genomes. Technical Report TR-CS2004-15, Department of Computer Science, University of New Mexico, 2004.

RNA Multiple Structural Alignment with Longest Common Subsequences*

Sergey Bereg[1] and Binhai Zhu[2]

[1] Department of Computer Science, University of Texas at Dallas
Richardson, TX 75083-0688, USA
besp@utdallas.edu
[2] Department of Computer Science, Montana State University
Bozeman, MT 59717-3880, USA
bhz@cs.montana.edu

Abstract. In this paper, we present a new model for RNA multiple sequence structural alignment based on the *longest common subsequence*. We consider both the off-line and on-line cases. For the off-line case, i.e., when the longest common subsequence is given as a linear graph with n vertices, we first present a polynomial $O(n^2)$ time algorithm to compute its maximum nested loop. We then consider a slightly different problem – the Maximum Loop Chain problem and present a factor-2 approximation which runs in $O(n^{2.5})$ time. For the on-line case, i.e., given m RNA sequences of lengths n, compute the longest common subsequence of them such that this subsequence either induces a maximum nested loop or the maximum number of matches, we present efficient algorithms using dynamic programming when m is small.

1 Introduction

In the study of noncoding RNA (*ncRNA*), it is well known that the corresponding genes are very active among genomic DNA. There are four such genes (polymers of nucleotides): A, C, G and U. Different from regular genes, ncRNAs fold directly into secondary and tertiary structures and the stability of the foldings are mainly determined by A-U, C-G and G-U bonds.

However, it is still not completely known how such a ncRNA folds into secondary and tertiary structures. One of the method is to take a multiple sequence of ncRNAs and investigate their common folding patterns or secondary structures [4, 19, 20]. In [4], it is proposed that the largest common nested linear subgraph of m given linear graphs (induced by m ncRNA sequences of length n) presents a solution for this problem. This problem is NP-complete and the authors presented an $O(\log^2 n)$ approximation for this problem [4].

In this paper, we follow the general methodology of [4]. However, we think that computing largest common nested linear subgraph cannot perfectly solve

* This research is partially supported by EPSCOR Visiting Scholar's Program and MSU Short-term Professional Development Program.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 32–41, 2005.

the problem in many situations. For example, if we have two ncRNA sequences: AGUU and CAGG, even though they induce the same largest common nested linear subgraph, the corresponding bonds and letters are completely different. (A letter cannot form a *bond* or *match* with a neighboring letter.)

The above idea forms the basis of our research. In this paper, we propose to use the Longest Common Subsequence (LCS) of m given ncRNA sequences as the basis to tackle this problem. We consider two general cases: off-line and on-line cases. In the off-line case, the LCS is already given and we want to find meaningful properties of such a LCS, namely, whether this LCS admits a special kind of fold. In the on-line case, we want to compute the LCS which admits certain kind of folding.

In general, the longest common subsequence of two sequences is not unique. Rick [17] developed an algorithm for finding *all* longest common subsequences. The number of longest common subsequences can be quite large. Greenberg [9] proved an exponential lower bound for the maximum number of distinct longest common subsequences of two sequences of length n. Therefore, in a lot of biological applications we believe that merely finding a longest common subsequence is not quite meaningful. In fact, finding a longest common subsequence satisfying a useful property is the goal of this paper. To the best of our knowledge, this has never done before, though there have been a lot of related work on identifying a (sub)string which is close to a set of given strings [13, 14] and close to a set of 'bad' strings and far from a set of 'good' strings [5].

In this paper, we mainly focus on three kinds of folding: maximum nested loop, maximum loop chains and maximum number of total matches. For the off-line case the problem is more of a graph theoretical one and we present both exact and approximation solutions. For the on-line case, the problem is NP-complete in general as computing LCS of multiple sequences, even without any other constraints, is NP-complete. We try to present efficient algorithms for cases when m is relatively small.

2 Preliminaries

Two characters (letters) $a, b \in \{A, C, G, U\}$ *match* or form a *bond* if $\{a, b\} = \{A, U\}$, or $\{a, b\} = \{C, G\}$, or $\{a, b\} = \{G, U\}$. Given a sequence $t = a_1 a_2 ... a_n$, $a_i \in \{A, C, G, U\}$, the corresponding linear graph $G(t)$ is defined as follows. The vertices of $G(t)$ are integers $1, 2, ..., n$ and there is an edge between i and $j (j > i + 1)$ if a_i and a_j match each other. For the obvious reason a_i cannot match a_{i+1} for $i = 1, 2, ..., n - 1$. In other words, there is no edge between i and $i + 1$ for $i = 1, 2, ..., n - 1$. This linear graph certainly characterizes the general folding possibilities of all the letters in t. In [7], a similar graph called *contact map graph* is also used for identifying protein structure similarity.

Given two edges e_1, e_2 in $G(t)$ and the intervals $I_1 = [a, b], I_2 = [c, d]$ spanned by them, we say e_1 intersects e_2 if exactly one of a, b lies on $[c, d]$ and vice versa. Therefore, if e_1 does not intersect e_2, then either I_1 and I_2 are disjoint or I_1 is contained in I_2 (assuming I_2 is longer). For the latter case we simply

denote $I_1 \subset I_2$ with the understanding that e_1 does not intersect e_2. A set of edges $e_1, e_2, ..., e_p$ in $G(t)$ form a *nested loop with depth p* if the intervals $I_1, I_2, ..., I_p$ spanned by $e_1, e_2, ...e_p$ is properly contained in one another, i.e., $I_1 \subset I_2 \subset \cdots \subset I_p$ (Figure 1 (1)).

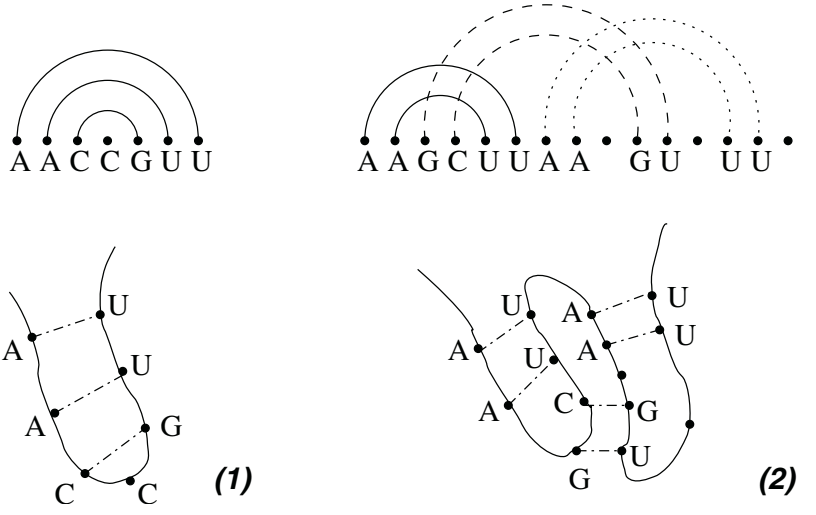


Fig. 1. An illustration of ncRNA folding with maximum loop and maximum loop chains.

Given a linear graph $G(t)$, two loops *overlap* if the edges in one loop L_1 intersect all the edges in the other loop L_2. Such an overlap is *legal* if no two edges from L_1, L_2 share the same vertex in $G(t)$. We define a *chain of loops* (or *loop chains*) as a set of loops $L_1, L_2, ..., L_w$ such that L_i overlaps with L_{i+1} but does not overlap with L_{i+x} for $i \leq w-1, x > 1$; moreover, each overlap is legal. The motivation behind this is that a chain of (relatively deep) loops provide a special kind of stable folding for a given ncRNA sequence (Figure 1 (2)).

In this paper, we propose to study several problems based on the Longest Common Subsequence (LCS). The LCS problem has been throughly studied in Hirschberg's PhD thesis [10]. Its application in computational biology dated back to 1960s [2, 3]. Some other applications of LCS in computational biology can be found in [11, 18]. Basically, for a set of m (m being a constant) sequences of length n, the corresponding LCS can be computed in $O(n^m)$ time. If m is not a constant, then the problem is NP-complete; moreover, if the alphabet is unbounded then it is difficult to find an approximation solution (in fact, as hard as approximating the Maximum Clique problem) [12].

3 The Off-Line Case: When the LCS Is Already Given

For the ncRNA multiple structural alignment problem, in general we want to compute a LCS with some additional constraints. In this section, we consider

the off-line case when a LCS of some ncRNA sequences is already computed. The first problem is based on the idea that the (maximum) deepest nested loop is likely to occur in ncRNA folding (Figure 1 (1)). The second problem is based on the idea that a chain of loops is likely to fold compactly with some specified regions (Figure 1 (2)).

3.1 The Maximum Nested Loop Problem

Given a sequence (which is the LCS of some ncRNA sequences) $t = a_1a_2...a_n, a_i \in \{A, C, G, U\}$, and the corresponding linear graph $G(t)$, compute the maximum or the deepest nested loop (MNL) in $G(t)$. We have the following theorem.

Theorem 1. *Given a sequence $t = a_1a_2...a_n, a_i \in \{A, C, G, U\}$, and the corresponding linear graph $G(t)$, the maximum nested loop can be computed in $O(n^2)$ time.*

Proof. For $1 \leq i \leq j \leq n$, let $t_{i,j}$ denote the sequence $a_i, a_{i+1}, \ldots, a_j$. Let S be a two-dimensional array where $S[i, j]$ is the maximum depth of a nested loop in the sequence $t_{i,j}$. The values of the array S can be computed as follows. For any $1 \leq i \leq n$, $S[i, i] = 0$.

Suppose that, for $1 \leq i < j \leq n$, a_i and a_j match. Then there is a maximum nested loop of $t_{i,j}$ that contains the edge (a_i, a_j). Thus $S[i, j] = S[i+1, j-1]+1$.

Suppose that a_i and a_j do not match. Then either a_i or a_j is not an endpoint of the outermost edge of a maximum nested loop of $t_{i,j}$. Thus, $S[i, j] = \max(S[i+1, j], S[i, j-1])$. We summarize all the cases in pseudo-code (Algorithm MNL).

The depth of maximum nested loop in the input sequence is $S[1, n]$. In order to compute the nested loop we store auxiliary arrays A and B such that $(A[i, j], B[i, j])$ is the outermost edge of a maximum nested loop of $t_{i,j}$. The values $A[i, j]$ and $B[i, j]$ can be updated at the time when $S[i, j]$ is updated.

The algorithm clearly takes $O(n^2)$ in the worst case. □

A slightly different $O(n^3)$ time result on loop matching in programming language research was known long time ago [16]. That result has been used in RNA folding [4]. Although the Maximum Nested Loop problem is slightly more easier to solve, in biology it could be a very important subroutine. In ncRNAs, A-U, C-G and G-U bonds almost always occur in a nested fashion [4]; so finding such maximum nested loop is very meaningful, at least it will allow biologists to try different alternatives in folding.

3.2 The Maximum Loop Chain Problem

In this subsection we investigate a slightly different problem. When a ncRNA sequence (with its corresponding linear graph) and a set of nested loops are given, we have the following problem of computing the *maximum loop chain*.

Algorithm MNL
for $l = 1$ to n
  for $i = 1$ to $n - l + 1$
    $j = i + l - 1$
    if $l = 1$ then
      $S[l, l] = 0$
    elseif a_i and a_j match then
      $S[i, j] = S[i + 1, j - 1] + 1$
      $A[i, j] = i$
      $B[i, j] = j$
    elseif $S[i + 1, j] > S[i, j - 1]$ then
      $S[i, j] = S[i + 1, j]$
      $A[i, j] = A[i + 1, j]$
      $B[i, j] = B[i + 1, j]$
    else
      $S[i, j] = S[i, j - 1]$
      $A[i, j] = A[i, j - 1]$
      $B[i, j] = B[i, j - 1]$

The **Maximum Loop Chain** problem: Given a ncRNA sequence $t = a_1 a_2 ... a_n$, $a_i \in \{A, C, G, U\}$, the corresponding linear graph $G(t)$ and a set of nested loops in $G(t)$, compute a loop chain out of these loops such that its size is maximized.

The *size* of a loop chain is the sum of depths of those loops in it. The motivation of this problem is that between two overlapping loops some edges might share the same vertices (i.e., *illegal*), which violates rules in ncRNA bonding (Figure 2). In the case of Figure 2, there is one node (corresponding to U) in the

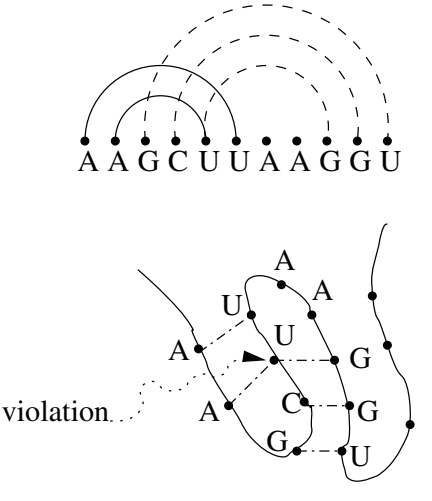


Fig. 2. A violation in ncRNA folding with two loop chains.

corresponding linear graph which is shared by two overlapping loops. Among the edges $(A, U), (U, G)$ we can only pick up one of them. So the problem is a matter of identifying the right loops and eliminating illegal edges in them.

It is not known yet whether this problem is NP-complete. We have the following approximation results.

Theorem 2. *The Maximum Loop Chain problem can be approximated in the following sense: in $O(n^{2.5})$ time a loop chain can be computed whose size is at least $\frac{1}{2}$ of the corresponding optimum.*

Given a set of nested loops we can construct a graph G' as follows: each node v_i corresponds to a nested loop L_i, there is an edge e_{ij} between two nodes v_i, v_j if their corresponding loops L_i, L_j overlap. Given two nested loops L_i, L_j with depths $|L_i|, |L_j|$ respectively, let $|L_i \cap L_j|$ be the number of nodes (letters) shared by edges in L_i, L_j. The weight of e_{ij} can be defined as $|L_i| + |L_j| - |L_i \cap L_j|$.

Then, the Maximum Loop Chain problem is to compute a path in G' such that between two neighboring nodes v_i and v_j, we need to throw away at most $|L_i \cap L_j|$ edges in either L_i or L_j (to obtain two new, smaller loops) such that eventually no two neighboring loops overlap illegally and the total number of edges in the loop chain is maximized. Notice that the weights on the edges in the final path corresponding to the maximum loop chain might not be the same as those initially in G'. It is very much a longest (heaviest) path problem with varying edge weights. We believe that this problem is NP-complete.

We will use the maximum weighted matching in G' to obtain an approximation solution for this problem. We claim that this provides at least $\frac{1}{2}$ number of bonds in the optimal loop chain. First assume that G' is also laid out from left to right on a line $\mathcal{L}$: the vertex in G' which corresponds to a loop touching the leftmost vertex in $G(t)$ is the leftmost on $\mathcal{L}$ and a tie is broken arbitrarily. The optimal loop chain then corresponds to a path with length k in G': $< v_{11}, v_{12}, ..., v_{1k} >$. If we take edges along this path at odd (even) positions (these two subsets of edges are disjoint), one subset of them contains at least half of the total bonds (edges in $G(t)$) in the optimal solution. Certainly the solution from the maximum weighted matching in G' is at least half of the total bonds (edges in $G(t)$) in the optimal solution for maximum loop chain.

Given a weighted graph with $|V|$ vertices and $|E|$ edges, the maximum weighted matching can be constructed in $O(|V|^3)$ time by Gabow [8], this was further improved to $O(\sqrt{|V|}|E|)$ time by Micali and Vazirani [15]. In our problem, G' clearly has $O(n)$ vertices and $O(n^2)$ edges as we are only interested in maximal nested loops. So the running time for computing the maximum weighted matching is $O(n^{2.5})$.

4 The On-Line Case: When the LCS Is Not Given

In this section, we study the problem when the LCS of a set of m ncRNA sequences are not given in advance. We study two versions of this general problem: LCSMNL and LCSBM. As computing LCS for multiple sequences is in general

NP-complete, both of these problems are NP-complete. We are interested in efficient solutions when m is relatively small. Recall that there might be too many LCS's for some given sequences, our goal is to identify a LCS with some useful property.

4.1 LCSMNL

We first consider the problem of computing the *longest common subsequence with maximum nested loop* (LCSMNL).

LCSMNL Problem. Given a set S of strings $s_1, s_2, \ldots, s_m$, each of length n, where $s_i = a_{i1}a_{i2}\ldots a_{in}$ and $a_{ij} \in \{A, C, G, U\}$, compute the longest common subsequence $s = b_1 b_2 \ldots b_K$ of $s_1, s_2, ..., s_m$ such that the maximum nested loop induced by s is at least L.

Theorem 3. *The* **LCSMNL** *problem can be solved in* $O(n^{m+2})$ *time. When* $m = 2$*, the problem can be solved in* $O(n^4)$ *time.*

Proof. We show the algorithm for $m = 2$ only. It is straightforward to extend it to general $m \geq 2$. We use a simplified notation $s_1 = a_1, a_2, \ldots, a_n$ and $s_2 = b_1, b_2, \ldots, b_n$. Let $a_{i,j}$, resp. $b_{i,j}$, denote the substring $a_i, a_{i+1}, \ldots, a_j$, resp. $b_i, b_{i+1}, \ldots, b_j$. We store four four-dimensional arrays L, D, A, B defined as follows. For $1 \leq i \leq j \leq n$ and $1 \leq k \leq l \leq n$, let $\Xi(i, j, k, l)$ denote the set of all longest common subsequences of $a_{i,j}$ and $b_{k,l}$. Then

- $L[i, j, k, l]$ is the length of the longest common subsequence of $a_{i,j}$ and $b_{k,l}$,
- $D[i, j, k, l]$ is the depth of the maximum nested loop induced by a sequence $\xi \in \Xi(i, j, k, l)$,
- $(A[i, j, k, l], B[i, j, k, l])$ is an outermost edge of a maximum nested loop induced by a sequence $\xi \in \Xi(i, j, k, l)$.

The items of the array $L[]$ can be computed in the same way as the computation of longest common subsequences. The value of $D[i, j, k, l]$ can be computed as in the pseudo-code shown in Algorithm LCSMNL. The theorem follows. □

4.2 LCSBM

Finally, we study the problem of computing a LCS which induces the maximum number of total matches, or *longest common subsequence with bounded matches* (LCSBM).

LCSBM Problem. Given a set S of strings $s_1, s_2, \ldots, s_m$, each of length n, where $s_i = a_{i1}a_{i2}\ldots a_{in}$ and $a_{ij} \in \{A, C, G, U\}$, compute the longest common subsequence $s = b_1 b_2 \ldots b_K$ of $s_1, s_2, ..., s_m$ such that the total number of matches among non-adjacent b_i and b_j is at least Q.

We assume that $m = 2$, and the algorithm can be easily extended to $m \geq 3$. We use a simplified notation $s_1 = a_1, a_2, \ldots, a_n$ and $s_2 = b_1, b_2, \ldots, b_n$. Let $b : \{A, C, G, U\} \rightarrow \{A, C, G, U\}$ be the mapping between a character and another

one such that they form a bond. Then, $b(A) = U, b(C) = G, b(G) = U$ and vice versa. Adding another character $x \in \{A, C, G, U\}$ to the end of a string, x induces a number of matches to its non-adjacent characters following the above setting.

Algorithm LCSMNL
// Initialize $L[..]$
for $i = 1$ to n
  for $j = i - 1$ to n
    for $k = 1$ to n
      $L[i, j, k, k - 1] = 0$
      $L[k, k - 1, i, j] = 0$
// Compute $L[..]$
for $i = 1$ to n
  for $j = i$ to n
    for $k = 1$ to n
      for $l = k$ to n
        if $a_j = b_k$ then
          $L[i, j, k, l] = L[i, j - 1, k, l - 1] + 1$
        else $L[i, j, k, l] = \max(L[i, j - 1, k, l], L[i, j, k, l - 1])$
// Compute $D[..]$
for $l_1 = 1$ to n
  for $i = 1$ to $n - l_1 + 1$
    $j = i + l_1 - 1$
    for $l_2 = 1$ to n
      for $k = 1$ to n
        $l = k + l_2 - 1$
        if $j = i$ or $k = l$ then
          $D[i, j, k, l] = 0$
        elseif $a_i \neq b_k$ then
          $D[i, j, k, l] = \max(D[i + 1, j, k, l], D[i, j, k + 1, l])$
        elseif $a_j \neq b_l$ then
          $D[i, j, k, l] = \max(D[i, j - 1, k, l], D[i, j, k, l - 1])$
        elseif a_i matches a_j then
          $D[i, j, k, l] = D[i + 1, j - 1, k + 1, l - 1] + 1$
        else
          $D[i, j, k, l] = D[i + 1, j - 1, k + 1, l - 1]$

Let $LCS[i, j]$ be the length of the longest common subsequence of the sequences $s_1 = a_1 a_2 \dots a_i$ and $s_2 = b_1 b_2 \dots b_j$. The array $LCS[i, j]$ can be computed in $O(n^2)$ time [1].

Let a, c, g and u be integers and x be any letter from $\{A, C, G, U\}$. A sequence s is called (a, c, g, u, x)-*sequence* if s contains a letters A, c letters C, g letters G, u letters U and the last letter of s is x.

We use 6-dimensional array $M[n, n, n, n, n, 4]$ whose elements are defined as follows. Let $1 \leq i, j, k \leq n$ and $x \in \{A, C, G, U\}$ and $l = LCS[i, j]$. Let

$0 \leq a, c, g \leq l$ be any integers such that $u = l - a - c - g \in [0, l]$. The value $M[i, j, a, c, g, x]$ is -1 if there is there is no (a, c, g, u, x)-sequence s of length l that is the common subsequence of $a_1, a_2, \ldots, a_i$ and $b_1, b_2, \ldots, b_j$ such that the last letter of s is x. If such a subsequence exists, then $M[i, j, a, c, g, x]$ is the maximum number of matches of s. The pseudo-code is listed as Algorithm LCSBM.

Algorithm LCSBM
Initialize $M[i, j, i_1, j_1, k_1, x]$ to -1 if $i \geq 1$ or $j \geq 1$ and to 0 if $i = 0$ or $j = 0$.
Compute $LCS[]$
for $i = 1$ to n
  for $j = 1$ to n
    $l = LCS[i, j]$
    for $a = 0$ to l // a is the number of A in LCS
      for $c = 0$ to $l - a$ // c is the number of C in LCS
        for $g = 0$ to $l - a - c$ // g is the number of U in LCS
          $u = l - a - c$ // u is the numbers of U in LCS
          if $a_i = b_j$ and a_i is counted at least one time in (a, c, g, u) then
            Let (a', c', g', u') be the the same numbers as (a, c, g, u)
                with one letter a_i removed
            for each $x \in \{A, C, G, U\}$
              if $M[i, j, a', c', g', x] \geq 0$ then
                Let z be the total matches induced by a_i
                    if added to a (a', c', g', u', x)-sequence
                $M[i, j, a, c, g, a_i] = \max(M[i, j, a, c, g, a_i], M[i, j, a', c', g', x] + z)$
          if $a_i \neq b_j$ then
            for each $x \in \{A, C, G, U\}$
              $M[i, j, a, c, g, x] = \max(M[i - 1, j, a, c, g, x], M[i, j - 1, a, c, g, x])$

Theorem 4. *The* **LCSBM** *problem can be solved in* $O(n^{m+3})$ *time. When* $m = 2$*, the problem can be solved in* $O(n^5)$ *time.*

5 Concluding Remarks

In this paper, we study several versions of the problem for RNA multiple structural alignment using a LCS model. There are several interesting open problems related to this work: (1) When a sequence t (say, the LCS of m RNA sequences) and a set of nested loops from the inducing linear graph are given, is the problem of computing Maximum Loop Chain NP-complete? (2) In [4], given a multiple number of linear graphs, each with n vertices, computing the maximum common non-intersecting subgraph was shown to be NP-complete. But the $O(\log^2 n)$ approximation factor is too high to make the result practically meaningful. Can it be further reduced?

References

1. T. Cormen, C. Leiserson, R. Rivest, C. Stein. *Introduction to Algorithms*, second edition, MIT Press, 2001.
2. M. Dayhoff. Computer aids to protein sequence determination. *J. Theoret. Biology*, **8(1)**:97-112, 1965.
3. M. Dayhoff. Computer analysis of protein evolution. *Scientific American*, **221(1)**:86-95, 1969.
4. E. Davydov and S. Batzoglu. A computational model for RNA multiple structural alignment. *Proc. 15th Ann. Symp. Combinatorial Pattern Matching,* LNCS 3109, pp. 254-269, 2004.
5. X. Deng, G. Li, Z. Li, B. Ma and L. Wang. A PTAS for distinguishing (sub)string selection. *Proc. ICALP'02*, pp. 740-751, 2002.
6. S.R. Eddy. Computational genomics of noncoding RNA genes. *Cell*, **109:**137-140, 2002.
7. D. Goldman, S. Istrail and C. Papadimitriou. Algorithmic aspects of protein structure similarity. *Proc. 40th Ann. Symp. Foundations of Computer Science (FOCS'99)*, pp. 512-522, 1999.
8. H. Gabow. An efficient implementation of Edmond's algorithm for maximum matching on graphs. *J. ACM*, **23(2):**221-234, 1976.
9. R. I. Greenberg. Bounds on the Number of the Longest Common Subsequence Problem. *CoRR cs.DM/0301030*, 2003.
10. D. Hirschberg. The longest common subsequence problem. *PhD Thesis*, Princeton University, 1975.
11. W.J. Hsu and M.W. Du. Computing a longest common subsequence for a set of strings. *BIT*, **24:**45-59, 1984.
12. T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM J. Comput.*, **24(5)**:1122-1139, 1995.
13. K. Lanctot, M. Li, B. Ma, S. Wang and L. Zhang. Distinguishing string selection problems. *Proc. 6th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pp. 633-642, 1999.
14. M. Li, B. Ma and L. Wang. Finding similar regions in many strings. *Proc. 31st ACM Symp. on Theory of Computing (STOC'99)*, pp. 473-482, 1999.
15. S. Micali and V. Vazirani. An $O(\sqrt{|V|}|E|^2)$ algorithm for finding maximum matching in general graphs. *Proc. 21st Ann. Symp. Foundations of Computer Science (FOCS'80)*, pp. 17-27, 1980.
16. R. Nussinov, G. Pieczenik, J. Griggs and D. Kleitman. Algorithms for loop matching. *SIAM J. Applied Math.*, **35:**68-82, 1978.
17. C. Rick. Efficient Computation of All Longest Common Subsequences. *Proc. 7th Scandinavian Workshop on Algorithm Theory (SWAT'00)*, pp. 407-418, 2000.
18. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. Molecular Biology*, **147:**195-197, 1981.
19. M. Zucker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, **9:**133-148, 1981.
20. M. Zucker. Computer prediction of RNA structure. *Methods in Enzymology*, **180:**262-288, 1989.

Perfect Sorting by Reversals*

Marie-France Sagot and Eric Tannier

INRIA Rhône-Alpes, Université de Lyon 1, France
{Eric.Tannier,Marie-France.Sagot}@inrialpes.fr

Abstract. In computational biology, gene order data is often modelled as signed permutations. A classical problem in genome comparison is to detect conserved segments in a permutation, that is, genes that are co-localised in several species, indicating that they remained grouped during evolution. A second largely studied problem related to gene order data is to compute a minimum scenario of reversals that transforms a signed permutation into another. Several studies began to mix the two problems, and it was observed that their results are not always compatible: often parsimonious scenarios of reversals break conserved segments. In a recent study, Bérard, Bergeron and Chauve stated as an open question whether it was possible to design a polynomial time algorithm to decide if there exists a minimum scenario of reversals that transforms a genome into another while keeping the clusters of co-localised genes together. In this paper, we give this polynomial algorithm, and thus generalise the theoretical result of the aforementioned paper.

1 Introduction

In computational biology, it is commonly accepted, using a parsimony argument, that if a group of homologous genes (that is genes having a common ancestry) is co-localised in two different species, then these genes were probably together in the common ancestor and were not later separated during evolution. The detection of such conserved clusters of homologous genes, also called *conserved segments*, has already been the subject of several algorithmic studies (see for instance [1, 8]).

In the theory of rearrangements, applying the parsimony principle means minimising the number of events in a reconstruction of possible evolutionary events between species. The algorithmics related to the rearrangements theory has also been intensively studied. The main results have been obtained on the problem of sorting by reversals [4, 7], which is a common event in evolution. The problem in this case concerns finding an optimal scenario of reversals, that is a shortest sequence of reversals that transforms one genome into the other.

A drawback of the methods developed so far for finding such parsimonious scenarios is that they do not respect the principle of conserved segments: despite

* This work is funded by the French program ACI "New interfaces of Mathematics: Mathematical and algorithmical aspects of biochemical and evolutionary networks", and by the INRIA coordinated action ARC "Integrated Biological Networks".

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 42–51, 2005.

the promising title of a former paper, *Common Intervals and Sorting by Reversals: A Marriage of Necessity* [3], it has indeed been noticed several times that in the case of reversals, the two criteria are not always compatible. A minimum rearrangement scenario may break conserved segments and then put them back together later again. A few studies [2, 6] began to mix the two principles. We go further in this direction, thus answering an open question mentioned in [2]. The question concerned the possibility of designing a polynomial time algorithm to decide whether there exists a minimum scenario of reversals that transforms a genome into another while keeping the clusters of co-localised genes together. In this paper, we give this polynomial algorithm, and thus generalise the theoretical result of the aforementioned paper.

We describe the usual model for dealing with gene order and orientation in the next section. In Section 3, we recall some basic facts about the structure of conserved segments of a permutation, as well as a padding operation described in [7] and adapted here to conserved segments. Finally, we give our main result and algorithm in Section 4.

2 Chromosomes as Signed Permutations

2.1 Generalities

Genome rearrangements such as reversals may change the order of the genes in a genome, and also the direction of transcription. We identify the genes with the integers $1, \ldots, n$, with a plus or minus sign to indicate their orientation. The order and orientation of genomic markers will be represented by a *signed permutation* of $\{1, \ldots, n\}$, that is, by a bijective function π over $[-n, n] \setminus \{0\}$ such that $\pi_{-i} = -\pi_i$, where $\pi_i = \pi(i)$.

To simplify exposition, we adopt the usual extension which consists in adding $\pi_0 = 0$, and $\pi_{n+1} = n + 1$ to the permutation. We therefore often define a signed permutation by writing $(0\ \pi_1\ \ldots\ \pi_n\ n+1)$. The *identity permutation* $(0\ 1\ \ldots\ n+1)$ is denoted by Id.

For all $i \in \{0, \ldots, n\}$, the pair $\pi_i\pi_{i+1}$ is called a *point* of π, and more precisely an *adjacency* if $\pi_i + 1 = \pi_{i+1}$ and a *breakpoint* otherwise. The number of points of a permutation π is denoted by $p(\pi)$, and the number of its breakpoints by $b(\pi)$.

The *reversal* of the interval $[i, j] \subseteq [1, n]$ $(i \leq j)$ is the signed permutation $\rho_{i,j} = (0\ \ldots\ i-1\ -j\ \ldots\ -i\ j+1\ \ldots\ n+1)$. Note that $\pi \cdot \rho_{i,j}$ is the permutation obtained from π by reversing the order and flipping the signs of the elements in the interval $[i, j]$:

$$\pi \cdot \rho_{i,j} = (\pi_0\ \ldots\ \pi_{i-1}\ \ -\pi_j\ \ldots\ -\pi_i\ \ \pi_{j+1}\ \ldots\ \pi_{n+1})$$

If $\rho_1, \ldots, \rho_k$ is a sequence of reversals, we say that it *sorts* a permutation π if $\pi \cdot \rho_1 \cdots \rho_k = Id$. The length of a shortest sequence of reversals that sorts π is called the *reversal distance* of π, and is denoted by $d(\pi)$. A shortest sequence of reversals sorting π is called a *parsimonious* sequence.

A *segment* of a permutation π is a set $\{|\pi_a|, \ldots, |\pi_b|\}$, with $1 \leq a < b \leq n$. The numbers π_a and π_b are the *extremities* of the segment. Two segments are

said to *overlap* if they intersect but one is not contained in the other. A reversal $\rho_{i,j}$ *breaks* $\{|\pi_a|, \ldots, |\pi_b|\}$ if $[i, j]$ and $[a, b]$ overlap. A sequence of reversals *breaks* a segment S if at least one reversal of the sequence breaks S.

2.2 The Breakpoint Graph

The breakpoint graph is a usual tool for dealing with signed permutations. It is present in almost every study on sorting by reversals. We use it intensively in the proofs of correctness of our method.

The *breakpoint graph* $BG(\pi)$ of a permutation π is a graph with vertex set V defined as follows: for each integer i in $\{1, \ldots, n\}$, let i^- and i^+ be two vertices in V; add to V the two vertices 0^+ and $(n+1)^-$. Observe that all vertex labels are non negative numbers, but for simplicity and to avoid having to use absolute values, we may later refer to vertex $(-i)^+$ (or $(-i)^-$): this is the same as vertex i^+ (or i^-).

The breakpoint graph of a signed permutation has sometimes been called the *diagram of desire and reality* due to the edge set E of $BG(\pi)$, which is the union of two perfect matchings of V, denoted by R, the *reality edges* and D, the *desire edges*:

- D contains the edges $i^+(i+1)^-$ for all $i \in \{0, \ldots, n\}$;
- R contains an edge for all $i \in \{0, \ldots, n\}$, from π_i^+ if π_i is non negative, and from π_i^- otherwise, to π_{i+1}^- if π_{i+1} is non negative, and to π_{i+1}^+ otherwise.

Reality edges define the permutation π (what you have), and desire edges define Id (what you want to have).

To avoid case checking, in the notation of an edge, the mention of the exponent $+$ or $-$ may be omitted. For instance, $\pi_i\pi_{i+1}$ is a reality edge, indicating nothing as concerns the signs of π_i and π_{i+1}.

It is easy to check that every vertex of $BG(\pi)$ has degree two (it has one incident edge in R and one in D), so the breakpoint graph is a set of disjoint cycles. By the cycles of a permutation π, we mean the cycles of $BG(\pi)$. The number of cycles of π is denoted by $c(\pi)$.

2.3 Conserved Segments

Let π be a signed permutation of $\{1, \ldots, n\}$, and $S = \{|\pi_a|, \ldots, |\pi_b|\}$ a segment of π, for $[a, b] \subseteq [1, n]$. Let $m = \min_{i \in [a,b]} |\pi_i|$ and $M = \max_{i \in [a,b]} |\pi_i|$.

The segment S is said to be *oriented* if there exist $i, j \in [a, b]$, such that π_i and π_j have different signs, and *unoriented* otherwise.

The segment S is said to be *sorted* if for all $i \in [a, b-1]$, the point $\pi_i\pi_{i+1}$ is an adjacency. A sorted segment is always unoriented. It is sorted *positively* if $\pi_a > 0$ and *negatively* if $\pi_a < 0$. In $\pi = (0\ -7\ \ 3\ -1\ \ 4\ \ 2\ \ 8\ -6\ -5\ \ 9)$, $\{6, 5\}$ is sorted negatively.

The segment S is said to be *conserved* if $M - m = b - a$. In $\pi = (0\ -7\ \ 3\ -1\ \ 4\ \ 2\ \ 8\ -6\ -5\ \ 9)$, $\{3, 1, 4, 2\}$ is conserved.

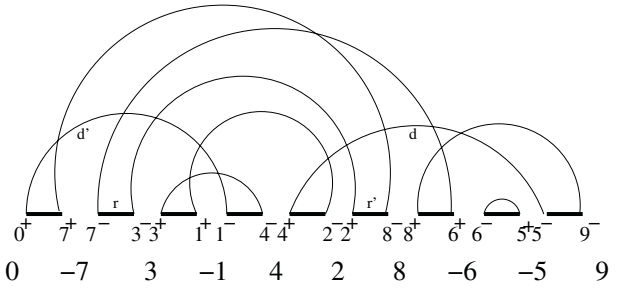


Fig. 1. The breakpoint graph of the permutation $(0\ -7\ 3\ -1\ 4\ 2\ 8\ -6\ -5\ 9)$. Reality edges are represented in bold, and desire edges are represented by thin lines.

The segment S is said to be *isolated* if it is conserved, either $\pi_a = m$ and $\pi_b = M$, or $\pi_a = -M$ and $\pi_b = -m$, and it is minimal, in the sense that the first and last point of the segment are breakpoints, and not adjacencies. In $\pi = (0\ -7\ -4\ 2\ -3\ -1\ 8\ -6\ -5\ 9)$, the segment $\{4, 2, 3, 1\}$ is isolated.

The segment S is said to be *highly conserved* if there exists a parsimonious sequence of reversals which does not break S.

A highly conserved segment is conserved, but the converse is not true. For example, in $(0\ -2\ -3\ 1\ 4)$, $\{2, 3\}$ is conserved but any parsimonious scenario breaks it.

An isolated segment is not always highly conserved. However, the permutations for which this is not the case are rare and irrelevant for our study, as we shall see in Section 2.4.

According to [2], we say that a sorting sequence of reversals is *perfect* if it breaks no conserved segment. If a permutation has a perfect parsimonious scenario, then all its conserved segments are highly conserved. The converse is however not true: for example, in $(0\ -3\ 4\ -1\ 2\ 5)$, both $\{1, 2\}$ and $\{3, 4\}$ are highly conserved, but any parsimonious sequence of reversals breaks one of them.

Perfect sorting sequences of minimum size have been studied in [6]. It is proved that given a permutation and a subset $\mathcal{S}$ of its conserved segments, it is NP-hard to compute the minimum scenario that does not break the segments of $\mathcal{S}$. The problem of finding a perfect sequence of reversals of minimum length is still open, to our knowledge. In [6], the following easy but fundamental lemma is presented.

Lemma 1. *If a sequence of reversals sorts a permutation and does not break a segment S, then there exists a sorting sequence of same size (with the same reversals), in which all the reversals contained in S (they sort S) are before all the other reversals (they sort outside S).*

The parsimonious scenario such that as few reversals as posible break some conserved segments is evoked in [2], but not solved. The authors study a special class of permutations for which there exists a perfect parsimonious scenario, and

the question is asked whether it is possible to decide in polynomial time, given a permutation, if there is a perfect parsimonious scenario that sorts it. In Section 4, we give this algorithm. Before that, we still need some preliminaries.

2.4 Sorting by Reversals

The main result about sorting by reversals is a theorem of Hannenhalli and Pevzner [7], which yielded the first polynomial algorithm to find a parsimonious sequence of reversals sorting any signed permutation.

We mention here a weaker version of this theorem, to avoid introducing notions which are useless for our purpose. One of the consequences of the general version of Hannenhalli and Pevzner's theorem is that it is possible to characterise the permutations for which all parsimonious sequences of reversals have to break some isolated segment. According to the standard vocabulary, they are the permutations that need a "hurdle merging". They can be characterised in this way.

Lemma 2. *[7] A permutation has an isolated segment which is not highly conserved if and only if it has at least three unoriented isolated segments A, B, C, such that either $A \subset B \subset C$, or $A \subset B$ and $C \cap B = \emptyset$.*

We call such permutations *fools*. They will obviously never have a perfect parsimonious scenario. We therefore start by assuming that the permutations we treat are not fools. It is easy to decide in linear time if a permutation is a fool or not (see for example [4]). We denote by $u(\pi)$ the number of unoriented isolated segments in a permutation π.

Theorem 1. *[7] Let π be a permutation but not a fool. Then $d(\pi) = p(\pi) - c(\pi) + u(\pi)$.*

This means that any reversal in a parsimonious scenario increases the number of cycles of the permutation ($p(\pi) = n + 1$ does not change after a reversal), except one (the first one) for each unoriented isolated segment. Each isolated segment is sorted separately (by definition, they do not overlap), and independently from the rest of the permutation.

3 Isolating Conserved Segments

As mentioned in the previous section, two isolated segments cannot overlap, and so each isolated segment is treated separately in any sorting algorithm. This is not immediately the case in general for conserved segments, but conserved segments of a permutation have a nested structure as well. This structure and a *padding* operation first described in [7] will allow to "isolate" conserved segments, and sort them independently from the rest of the permutation when it is possible.

3.1 The Structure of Conserved Segments

We recall basic facts about the structure of conserved segments, that are useful for our purpose. The reader may refer to [5] for a general presentation on modular structures.

A conserved segment is called *strong* if it does not overlap any other conserved segments. By definition, the family of strong conserved segments is nested. Strong segments can be of two types. Suppose all non trivial strong conserved segments strictly contained in a strong segment S have been contracted into a single representative number, and the result of these contractions is $\{a, \ldots, b\}$. If $\{a, \ldots, b\}$ is an increasing or decreasing sequence of consecutive numbers, S is called *linear*. If no proper subset of $\{a, \ldots, b\}$ is conserved, S is called *prime*.

Lemma 3. *If S is a strong segment, then it is either linear, or prime.*

We sort each strong segment independentely, assuming that all the strong segments strictly included in it are already sorted (we start by the inclusionwise minimal ones).

3.2 Padding a Permutation

Isolating a conserved segment is done by an operation called *padding* that is described in [7], where it is used to transform a permutation into a simpler one, with equivalent properties. We show that it can be used to deal with conserved segments as well. To begin with, we want to be able to add elements to a permutation without changing the existing indices. To do so, we deal with "generalised" signed permutations in the sense that the permutations will be bijective functions from a set of indices to a set of values, both being ordered sets of reals instead of integer numbers. For example, $(0 \quad 3.5 \quad -3 \quad 1 \quad 2 \quad 4)$ is a generalised permutation, where $\pi_0 = 0$, $\pi_{0.5} = 3.5$, $\pi_1 = -3$, $\pi_2 = 1$, $\pi_3 = 2$, $\pi_4 = 4$.

A *padding* of a permutation π consists in adding an index k such that $i < k < i + 1$, for some existing index i, and its image through π, such that $j < |\pi_k| < j + 1$, for some existing value j (π_k may be positive or negative). For example, if $\pi = (0 \quad -3 \quad 1 \quad 2 \quad 4)$ is a signed permutation over $\{0, 1, 2, 3, 4\}$, $\pi' = (0 \quad 3.5 \quad -3 \quad 1 \quad 2 \quad 4)$, is a padding of π with $0 < k < 1$ and $3 < |\pi_k| < 4$. The resulting generalised permutation π' has the same breakpoint graph as π, except that the two edges $r = \pi_i\pi_{i+1}$ and $d = j(j + 1)$ are now each split into two edges r_1, r_2 and d_1, d_2. Examples of padding are shown in Figures 2 and 3.

A padding is said to be *safe* if the resulting permutation π' has one cycle more than π, and no new unoriented isolated segment, that is, according to Theorem 1, if $d(\pi') = d(\pi)$. Any sequence sorting π' also sorts π (just ignore the added element). If the transformation is safe, a parsimonious scenario for π' will therefore provide a parsimonious scenario for π. By extension, a sequence of paddings is *safe* if the resulting permutation π' satisfies $d(\pi') = d(\pi)$.

Let $S = \{|\pi_a|, \ldots, |\pi_b|\}$ ($[a, b] \subseteq [1, n]$) be a conserved segment, $M = \max_{i\in[a,b]} |\pi_i|$, and $m = \min_{i\in[a,b]} |\pi_i|$. We say that S has a *positive padding* if it

is safe to pad it with an index k_1, such that $a-1 < k_1 < a$, and $m-1 < \pi_{k_1} < m$, and then with an index k_2, such that $b < k_2 < b+1$, and $M < \pi_{k_2} < M+1$. We say that S has a *negative padding* if it is safe to pad it with an index k_1, such that $a-1 < k_1 < a$, and $M < -\pi_{k_1} < M+1$, and then with an index k_2, such that $b < k_2 < b+1$, and $m-1 < -\pi_{k_2} < m$.

For example, in Figure 2, there is a negative padding of the conserved segment $\{3, 1, 4, 2\}$ in the permutation $(0\ -7\ \ 3\ -1\ \ 4\ \ 2\ \ 8\ -6\ -5\ \ 9)$. There is no positive padding of the same segment, as shown later in Figure 3.

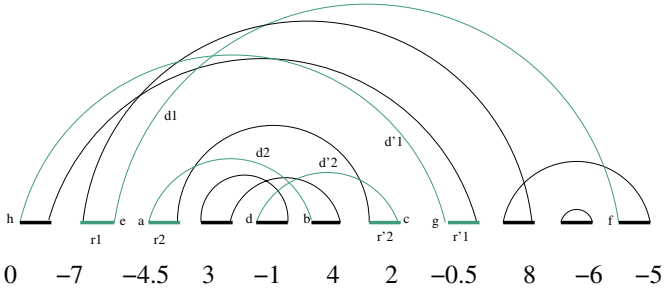


Fig. 2. A negative padding of segment $\{3, 1, 4, 2\}$ in the permutation $(0\ -7\ \ 3\ -1\ \ 4\ \ 2\ \ 8\ -6\ -5\ \ 9)$. Note that there are four cycles in the breakpoint graph, and no unoriented isolated segment. The segment $\{4.5, 3, 1, 4, 2, 0.5\}$ is isolated, but oriented. There was two cycles in $(0\ -7\ \ 3\ -1\ \ 4\ \ 2\ \ 8\ -6\ -5\ \ 9)$, so the padding is safe.

After a padding of a segment S, positive or negative, $S \cup \{\pi_{k_1}, \pi_{k_2}\}$ is isolated. The number of cycles of the breakpoint graph increases by two since the two paddings are safe. The proof of the following lemma is not included here, it is an easy verification.

Lemma 4. *A segment is highly conserved if and only if it has a padding (positive or negative).*

We now have the possibility to identify strong conserved segments, and test whether they are highly conserved or not. The remaining difficulty is to choose between a positive and a negative padding when both are possible.

4 Perfect Parsimonious Sequences of Reversals

As noticed in [6], the main difficulty in finding perfect sequences of reversals of minimum length (among all perfect sequences) is that it is sometimes impossible to decide whether to sort a particular segment positively or negatively. We shall see that in the case of parsimonious scenarios, this choice is constrained by the data.

We denote by $d_+(S)$ the minimum number of reversals needed to sort a conserved segment S positively, and $d_-(S)$ the minimum number of reversals

needed to sort it negatively. Of course, $|d_+(S) - d_-(S)| \leq 1$, because if it is sorted in one direction, then one reversal is sufficient to have it sorted in the other. If $d_+(S) = d_-(S)$, the segment S is called *neutral*.

If there is a perfect parsimonious scenario, any conserved segment is highly conserved, so has a safe padding from Lemma 4. However, the converse is not true. The possibility of designing a simple algorithm to decide the existence of a perfect parsimonious scenario is given by the following lemma.

Lemma 5. *If a segment is neutral, then it cannot have both a positive and a negative padding.*

Proof. Let $[a, b] \subseteq [1, n]$, such that $S = \{|\pi_a|, \ldots, |\pi_b|\}$ is a neutral conserved segment. Let $M = \max_{i\in[a,b]} |\pi_i|$, and $m = \min_{i\in[a,b]} |\pi_i|$. Suppose S has a negative padding, call π^- the resulting permutation. This means it is safe to pad S with an index k_1, such that $a - 1 < k_1 < a$, and $M < -\pi^-_{k_1} < M + 1$, and an index k_2, such that $b < k_2 < b + 1$, and $m - 1 < -\pi^-_{k_2} < m$. Let r and d be the reality and desire edges deleted after the padding with the index k_1, and r' and d' the reality and desire edges deleted after the padding with the index k_2 (see Figures 1 and 2 for an example). Suppose S has also a positive padding, and call π^+ the resulting permutation. This means that it is safe to pad S with the index k_1, with $m - 1 < \pi^+_{k_1} < m$, and the index k_2, with $M < \pi^+_{k_2} < M + 1$. The deleted edges are the same ones, except that r and d' are deleted after the padding with k_1, and r' and d are deleted after the padding with k_2.

As both paddings are safe, the number of cycles has to increase for each padding operation. We therefore have that r and d belong to the same cycle of π, as well as r' and d', r and d', r' and d. So r, d, r' and d' all belong to the same cycle in the breakpoint graph of π. Observe that because S is conserved, r, d, r' and d' are the only edges that have one extremity with a label inside S, and the other outside S.

In π^-, d and d' are both replaced by two edges, say d_1, d_2, and d'_1, d'_2. One edge among the two has its extremities with labels inside S, and the other outside S. As in Figure 2, say $d_2 = ab$, and $d'_2 = cd$ have their extremities with labels inside S.

In π^+, the edges which replace d and d' and have their extremities with labels inside S are ad and cb (see Figure 3 for an example).

Recall that d, d', r, r' are the only edges affected by the paddings, the remaining of the breakpoint graph is unchanged. So if ab and cd belong to different cycles in π^-, then ad and cb belong to the same cycle in π^+, and vice-versa. In this case however, the segment S would not be neutral as $d_-(S) \neq d_+(S)$ from Theorem 1. Since S is neutral, the edges ab and cd have to belong to the same cycle both in π^- and in π^+. It is the case in the example of Figures 2 (for π^-) and 3 (for π^+).

We now repeat the argument for the edges that have their extremities with labels outside S, that is d_1, d'_1. In π^-, let $d_1 = ef$ and $d'_1 = gh$. Then in π^+, $d_1 = eh$ and $d'_1 = gf$. If d_1 and d'_1 are in different cycles in π^-, they are in the same cycle in π^+, and vice-versa. We therefore have that either $c(\pi^-) = c(\pi)+1$,

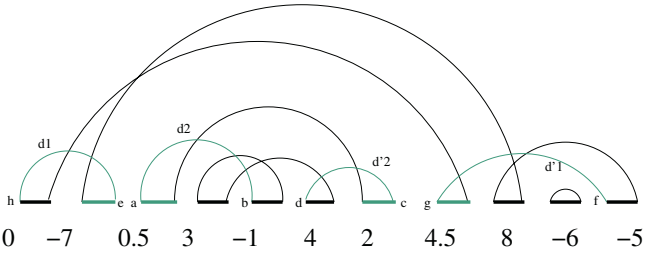


Fig. 3. An attempt of a positive padding of the segment $\{3, 1, 4, 2\}$ in the permutation $(0\ -\ 7\ \ 3\ -1\ \ 4\ \ 2\ \ 8\ -6\ -5\ \ 9)$. The graph is almost the same as for the negative padding. However edges are changed from ab and cd to ad and bc, so in one case there are two cycles outside the segment while in the other case there is only one such cycle. This explains why both paddings are impossible.

or $c(\pi^+) = c(\pi) + 1$, and one the paddings is not safe, because in this case, either $d(\pi^+) > d(\pi)$, or $d(\pi^-) > d(\pi)$. This is what happens in Figure 3, where $d(\pi^+) > d(\pi^-) = d(\pi)$.

As a consequence, a neutral segment cannot have both a positive and negative padding. □

The principle of the algorithm follows immediately.

Theorem 2. *Given a permutation π, it is possible to design in polynomial time a perfect parsimonious sequence of reversals sorting π if one exists.*

Proof. We apply the usual techniques to sort oriented isolated segments by reversals. We do not describe this in detail. One can see for instance [9] for a fast method to do so.

Let π be an arbitrary permutation. We first check if it is not a fool (see Lemma 2). If it is, there is no perfect parsimonious sorting sequence.

We now treat each isolated segment separately, starting with the inclusion-wise minimal ones, up to the whole permutation, or stopping when a contradiction is found.

- Sorting an oriented isolated segment

Suppose first that the considered isolated segment I (possibly $\{0, \ldots, n+1\}$) is oriented. Let now S be any inclusion-wise minimal strong conserved segment inside I. We try both paddings of S to see if it is highly conserved. If S is not highly conserved, then there is no perfect parsimonious sequence. If both paddings exist, then by Lemma 5, the segment S is not neutral. In this case, we choose the positive padding if $d_+(S) < d_-(S)$, and the negative one otherwise. If $d_+(S) = d_-(S)$, the choice of the padding is constrained. In every case, the permutation is padded with two values π_{k_1} and π_{k_2}, and $S \cup \{\pi_{k_1}, \pi_{k_2}\}$ is now isolated. By Lemma 3, either no proper subset of S is conserved, or all its proper subsets are conserved. We examine the two cases separately.

In the first case, we sort S with, for example, the method of [9], for isolated segments. This preserves all conserved segments because there is none inside S, and the way S is sorted does not affect the remaining of the permutation.

In the second case, the only allowed reversals are the reversals of singletons (the reversals of the whole interval is not an admitted operation, since otherwise the opposite padding is chosen). The reversals of singletons never overlap, so they may be applied in any order. If the padding is positive, we reverse all negative numbers, and if the padding is negative, we reverse all the positive numbers. At the end, the segment S is sorted.

We apply the same method to all the strong segments, starting with inclusion-wise minimal ones, up to the segment I itself.

- *Sorting an unoriented isolated segment*

Let us suppose now that I is unoriented. The first reversal will make it oriented, and then the aforementioned method ("Sorting an oriented isolated segment") is applied. To orient I, we choose a reversal that does not break any conserved segment, nor decreases $c(\pi)$. Every such reversal has to be tried. There are at most $|I|^2$ of them, and they are applied only at the first step, so this operation yields a polynomial algorithm.

- *Conclusion*

We have seen how to sort an isolated segment. All are sorted the same way, and separately. If there is a perfect parsimonious sequence sorting π, then the algorithm produces it, because the way an isolated segment is sorted never affects the permutation outside the segment.

This method therefore decides if there is a perfect parsimonious sequence sorting π in polynomial time. □

References

1. Beal M.-P., Bergeron A., Corteel S., Raffinot, M., "An Algorithmic View of Gene Teams", *Theor. Comput. Sci.* 320(2-3):395-418, 2004.
2. Bérard S., Bergeron A., Chauve C., "Conserved structures in evolution scenarios", 2nd RECOMB Comparative Genomics Satellite Workshop, to appear in *Lecture Notes in BioInfomatics*, 2004.
3. Bergeron A., Heber S., Stoye J., "Common Intervals and Sorting by Reversals: A Marriage of Necessity", *Bioinformatics*, 1:1-10, 2002.
4. Bergeron A., Mixtacki J., Stoye J., "The inversion distance problem", in *Mathematics of evolution and phylogeny* (O. Gascuel Ed.) Oxford University Press, 2005.
5. Bui Xuan B. M., Habib M., Paul C., "From permutations to Graph Algorithms", *Research Report* LIRMM RR-05021, 2005.
6. Figeac M., Varré J.-S., "Sorting by reversals with common intervals", Proceedings of WABI 2004, *Lecture Notes in Computer Science*, vol. 3240, 26-37, 2004.
7. Hannenhalli S., Pevzner P. , "Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals", *Proceedings of the 27th ACM Symposium on Theory of Computing*, 178-189, 1995.
8. Heber S., Stoye J., "Finding all Common Intervals of k Permutations", Proceedings of CPM 2001, *Lecture Notes in Computer Science*, vol. 2089, 207-218, 2001.
9. Tannier E., Bergeron A., Sagot M.-F., "Advances on Sorting by Reversals", to appear in *Discrete Applied Mathematics*, 2005.

Genome Rearrangements with Partially Ordered Chromosomes

Chunfang Zheng[1] and David Sankoff[2]

[1] Department of Biology
University of Ottawa, Canada K1N 6N5
czhen033@uottawa.ca
[2] Department of Mathematics and Statistics
University of Ottawa, Canada K1N 6N5
sankoff@uottawa.ca

Abstract. Genomic maps often do not specify the order within some groups of two or more markers. The synthesis of a master map from several sources introduces additional order ambiguity due to markers missing from some sources. We represent each chromosome as a partial order, summarized by a directed acyclic graph (DAG), to account for poor resolution and of missing data. The genome rearrangement problem is then to infer a minimum number of translocations and reversals for transforming a set of linearizations, one for each chromosomal DAG in the genome of one species, to linearizations of the DAGs of another species. We augment each DAG to a directed graph (DG) in which all possible linearizations are embedded. The chromosomal DGs representing two genomes are combined to produce a single bicoloured graph. From this we extract a maximal decomposition into alternating coloured cycles, determining an optimal sequence of rearrangements. We test this approach on simulated partially ordered genomes.

1 Introduction

1.1 Formalizing Genomes and Their Evolutionary Mechanisms

The genetic structure of a genome can be modeled as a set $\chi = \{\chi_1, \cdots, \chi_k\}$ of $k \geq 1$ chromosomes, where each chromosome χ_i consists of $n_i > 0$ genes or other genetic markers, signed and totally ordered: e.g., $g_1 < \cdots < g_{n_i}$, also written simply as $g_1 \cdots g_{n_i}$, where each g is a positive or negative integer, and where each g appears only once in all of χ. Without loss of generality, we may assume each integer between 1 and $n = n_1 + \cdots + n_k$ appears exactly once in χ, either with a plus or minus sign. n is the size of the genome. An example of a genome of size 8 is thus $\{\chi_1, \chi_2, \chi_3\}$, where $\chi_1 = 5\ -1\ -4, \chi_2 = 7\ 8\ 3$, and $\chi_3 = -2\ 6$. Henceforward we will use the term gene to refer to either genes or genetic markers of any kind. The order relation abstracts the position of the gene on the linear chromosome, and the sign carries information about which of the two DNA strands the gene is located.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 52–62, 2005.

Genome evolution can be modeled by the transformation of a genome χ of size n to a genome ψ of the same size, by means of reversals within a chromosome and translocations between chromosomes. A reversal transforms any contiguous part of an order to its reverse order, changing the polarity of all genes in its scope, e.g., $g\ h\ i\ j\ k \rightarrow g - j - i - h\ k$ is a reversal of the "segment" $h\ i\ j$. A (reciprocal) translocation exchanges any prefixes of two chromosomes (or equivalently, any suffixes of two chromosomes) or the reversed prefix of one with the reversed suffix of the other, for example $g\ h\ i, x\ y\ z \rightarrow g\ h\ y\ z, x\ i; g\ h\ i, x\ y\ z \rightarrow g - x, -i - h\ y\ z$. There are two special cases: $g\ h\ i, x\ y\ z \rightarrow g\ h\ i\ x\ y\ z; g\ h\ i\ x\ y\ z \rightarrow g\ h\ i, x\ y\ z$ where a null prefix of one chromosome $x\ y\ z$ is exchanged with the largest prefix of the other $g\ h\ i$ (chromosome fusion) and where a null chromosome translocates with $g\ h\ i\ x\ y\ z$ (chromosome fission), respectively. To make biological sense, since a chromosome does not change its nature when it is moved around in space, a reversal of an entire chromosome is considered to leave it unchanged: $g_1 \cdots g_{n_i} = -g_{n_i} \cdots - g_1$.

The Hannenhalli-Pevzner algorithms [2, 3] for comparing genomes, and their improvements (e.g., [1], [4]), infer $d(\chi, \psi)$, the smallest number of reversals and translocations necessary to transform genome χ into ψ, as well as a sequence of such operations that actually achieves this minimum.

1.2 Partially Ordered Genomes

The representation of a genome as a set of totally ordered chromosomes must often be weakened in the case of real data, where mapping information only suffices to partially order the set of genes on a chromosome. The concepts and methods of genome rearrangement, however, pertain only to totally ordered sets of genes or markers, and are meaningless in the context of partial orders.

Our approach is to extend genome rearrangement theory to the more general context where all the chromosomes are general DAGs rather than total orders [5]. The use of DAGs reflects uncertainty of the gene order on chromosomes in the genomes of most advanced organisms. This may be due to lack of resolution, where several genes are mapped to the same chromosomal position, to missing data from some of the datasets used to compile a gene order, and/or to conflicts between these datasets.

We construct the chromosomal DAGs for each species from two or more incomplete data sets, or from a single low-resolution data set. The frequent lack of order information in each data set, due to missing genes or missing order information, is converted into parallel subpaths within each chromosomal DAG in a straightforward manner.

Outright conflicts of order create cycles that must be broken to preserve a DAG structure. We suggest a number of reasonable alternative conventions for breaking cycles. This is not the focus of our analysis, however; whatever convention is adopted does not affect our subsequent analysis.

The rearrangement problem is then TO INFER A TRANSFORMATION SEQUENCE (TRANSLOCATIONS AND/OR REVERSALS) FOR TRANSFORMING A SET OF LINEARIZATIONS (TOPOLOGICAL SORTS), ONE FOR EACH CHROMOSOMAL

DAG in the genome of one species, to a set of linearizations of the chromosomal DAGs in the genome of another species, minimizing the number of translocations and reversals required. To do this, we embed the set of all possible linearizations in each DAG by appropriately augmenting the edge set, so that it becomes a general directed graph (DG). We combine the two sets of chromosomal DGs representing two genomes to produce a single large bicoloured graph from which we extract a maximal decomposition into alternating coloured cycles, so that a Hannenhalli-Pevzner type of procedure can then generate an optimal sequence of rearrangements. We focus here on obtaining the cycle decomposition; this is equivalent to optimally linearizing the partial orders, so that finding the rearrangements themselves can be done using the previously available algorithms.

2 Gene Order Data

2.1 The Methodological Origins of Incomplete Maps

Maps of genes or other markers produced by recombination analysis, physical imaging and other methods, no matter how highly resolved, inevitably are missing some (and usually most) genes or markers and fail to order some pairs of neighbouring genes with respect to each other. Even at the ultimate level of resolution, that of genome sequences, the application of different gene-finding protocols usually gives maps with different gene content.

Moreover, experimental methodologies and statistical mapping procedures inevitably give rise to some small proportion of errors, two neighbouring genes incorrectly ordered, a gene mapped to the wrong chromosome, a gene incorrectly named or annotated. However it is not these errors we focus on in this paper, but the more widespread issues of lack of resolution and genes missing from a map. These should not be considered errors; they are normal and inherent in all ways of constructing of a map except for highly polished genome sequencing with accurate gene identification (something that has not yet been achieved in the higher eukaryotes, even for humans).

2.2 Simulating Incomplete Maps of Pairs of Two Related Genomes

How incomplete maps arise may perhaps be best understood through a description of how we simulate them.

Simulating the Genomes. For a given n, we pick a small integer k, as well as positive $n_1, \cdots, n_k$ with the constraint $n = n_1 + \cdots + n_k$. Then we define

$$\chi = \{m_1 \cdots n_1, m_2 \cdots n_2, \cdots, m_k \cdots n_k\}, \tag{1}$$

where $m_1 = 1, n_k = n$ and the remaining $m_i = n_{i-1} + 1$. It is well known that the genes in two genomes being compared through translocation and reversal

distance may always be relabeled in so that in one of the genomes they have they all have positive sign and have the form in (1).

To obtain the second genome ψ, we perform r reversals distributed at random among the k chromosomes of χ, reversing randomly chosen segments, and t reciprocal translocations between random pairs of chromosomes, exchanging randomly sized prefixes or suffixes, plus f_1 fusions and f_2 fissions. The operations are performed in random order, each one applied to the transformed genome produced by the preceding operation.

The non-negative parameters $n, k, n_1, \cdots, n_k, r, t, f_1$ and f_2 are specified in advance, as is the random choice procedure, depending on the kind of genomes we wish to model and compare. For $n = 25, k = 4, n_1 = 7, n_2 = 8, n_3 = 5, n_4 = 5, r = 4, t = 4, f_1 = 0$ and $f_2 = 1$, (2) shows genome χ and an example of ψ.

$$
\begin{array}{l}
\{1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7,\\
=\ \ 8\ \ 9\ 10\ 11\ 12\ 13\ 14\ 15,\\
16\ 17\ 18\ 19\ 20,\\
21\ 22\ 23\ 24\ 25\}
\end{array}
\qquad
\begin{array}{l}
\{-17\ -3\ \ -2\ \ -1\ \ 18\ -5\ -4\ -16,\\
=\ \ \ 8\ -13\ -12\ -11\ -22\ -21,\\
\ \ -7\ \ -20\ \ 10\ \ -24\ -23\ \ 25,\\
\ \ 14\ \ 15,\\
\ \ 9\ -19\ \ 6\}
\end{array}
\tag{2}
$$

Simulating the Maps. Once we have obtained simulated genomes χ and ψ, we fix probabilities p_{missing} and p_{group}, and numbers of data sets N_χ and N_ψ.

For each of the two genomes we construct each of the N_χ or N_ψ data sets independently, as follows. For each chromosome, each gene on the chromosome except the last one is submitted to a grouping event with probability p_{group}, which determines whether or not the gene position can be distinguished from that of the next gene. Then each gene is submitted independently to a deletion event with probability p_{missing}, conditioned on the event that the gene cannot be deleted from all the datasets. Note that if a gene g_1 is grouped with the next gene g_2, which is subsequently deleted, and if g_2 was not itself grouped with the next gene g_3, then g_1 is not grouped with g_3 in the data set.

Note that this procedure cannot produce two data sets on the same genome with conflicting order relations ($a < b$ in one, $b < a$ in the other), nor with the same gene on two different chromosomes. Three data sets produced from the genomes in (2) are shown in (3), with boxes around unresolved groups of genes.

$$
\begin{array}{l}
\{\ 1\ \ 2\ \ \boxed{4\ 5}\ \ 7,\\
=\ \ \ 8\ 10\ 11\ 15,\\
\ \ 16\ 17\ 20,\\
\ \ 22\ 23\ 25\}
\end{array}
\qquad
\begin{array}{l}
\{\ \boxed{2\ 3}\ \ 6\ \ 7,\\
\ \ 8\ \ 9\ 12\ 14,\\
\ \ 17\ 18\ 19,\\
\ \ 21\ 24\ 25\}
\end{array}
\qquad
\begin{array}{l}
\{\ 2\ \ 4\ \ \ 6,\\
\ 9\ 11\ \boxed{12\ 13}\ \boxed{14\ 15},\\
\boxed{16\ 18}\ \ 20,\\
\qquad \boxed{21\ 23}\ 24\}
\end{array}
\tag{3}
$$

$$
\begin{array}{l}
\{\ -17\ \ -2\ \ \ 18\ \ -4,\\
=\ \boxed{-13\ -12\ -11}\ \boxed{-22\ -21},\\
\ \ \ -7\ \boxed{-20\ \ 10\ \ -24\ -23}\ \ 25,\\
\ \ \ 14\ \ \ 15,\\
\ \ \ \ 9\ -19\}
\end{array}
\qquad
\begin{array}{l}
\{\boxed{-17\ \ -3\ \ -2}\ \ -1\ \ \boxed{-5\ \ -4\ -16},\\
\ \ \ \ 8\ \ \boxed{-13}\ \boxed{-12\ \ -11\ -22},\\
\ \ \boxed{-7\ \ -20\ \ \ 10}\ \ \boxed{-24\ \ \ 25},\\
\ \ \ 14\ \ \ 15,\\
\ \ -19\ \ \ 6\}
\end{array}
\qquad
\begin{array}{l}
\{\ \boxed{-2\ \ \ 18}\ \ -5\ \ -16,\\
\ \ \ \ 8\ \boxed{-13\ \ -11}\ \ -21,\\
\ \ -20\ \ 10\ \ -24\ \ \ -23,\\
\ \ \ 14\ \ \ 15,\\
\ \ \ \boxed{9\ \ \ 6}\}
\end{array}
$$

3 Constructing the Chromosomal DAGs

A linear map of a chromosome that has several genes or markers at the same position π, because their order has not been resolved, can be reformulated as a partial order, where all the genes before π are ordered before all the genes at π and all the genes at π are ordered before all the genes following π, but the genes at π are not ordered amongst themselves. We call this procedure **make_po**.

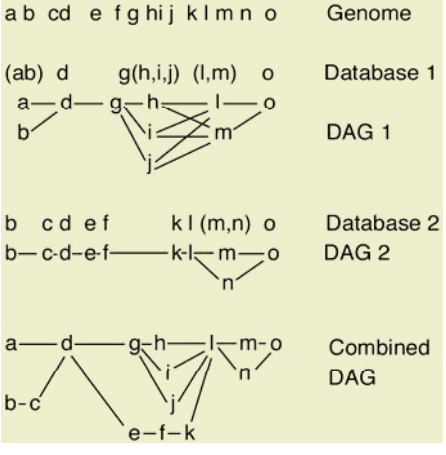


Fig. 1. Construction of DAGs from individual databases each containing partial information on genome, due to missing genes and missing order information, followed by construction of combined DAG representing all known information on the genome. All edges directed from left to right.

For genomes with two or more gene maps constructed from different kinds of data or using different methodologies, there is only one meaningful way of combining the order information on two (partially ordered) maps of the same chromosome containing different subsets of genes. Assuming there are no conflicting order relations ($a < b, b < a$) nor conflicting assignments of genes to chromosomes among the data sets (as in the data sets on our simulated genomes), for each chromosome we simply take the union of the partial orders, and extend this set through transitivity. This procedure is **combine_po**.

All the partial order data on a chromosome can be represented in a minimal DAG whose vertex set is the union of all gene sets on that chromosome in the contributing data sets, and whose edges correspond to just those order relations that cannot be derived from other order relations by transitivity. The outcome of this construction, **dagger**, is illustrated in Figure 1.

In real applications, different maps of the same genome do occasionally conflict, either because $b < a$ in one data set while $a < b$ in the other or because a gene is assigned to different chromosomes in the two data sets. There are a variety of possible ways of resolving order conflicts or, equivalently, of avoiding

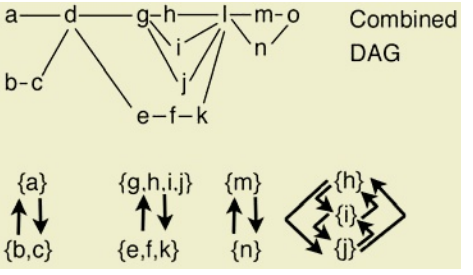


Fig. 2. Edges added to DAG to obtain DG containing all linearization as paths (though not all paths in the DG are linearizations of the DAG!). Each arrow represents a set of directed edges, one from each element in one set to each element of the other set.

any cycles in the construction of the DAG. One way is to delete all order relations that conflict with at least one other order relation. Another is to delete a minimal set of order relations so that all conflicts can be resolved. Still another is to ignore a minimum set of genes that will accomplish the same end. The latter method also resolves conflicts due to gene assignment to different chromosomes. Any of these approaches, or others, which we denote by the generic routine name **resolve**, will produce results appropriate for our subsequent analysis.

4 The DG Embedding of Topological Sorts

A DAG can generally be linearized in many different ways, all derivable from a topological sorting routine. All the possible adjacencies in these linear sorts can be represented by the edges of a directed graph (DG) containing all the edges of the DAG plus two edges of opposite directions between all pairs of vertices, which are not ordered by the DAG. This is illustrated in Figure 2. The routine for constructing this graph is **dgger**.

5 The Algorithm

5.1 Background

Before discussing our algorithm for comparing DGs derived from our DAG representations, we review the existing technology for the special case when the DAG and the associated DG represent a total order, which is the traditional subject of computational comparative genomics.

Hannenhalli and Pevzner [2] showed how to find a shortest sequence of reversals and translocations that transform one genome χ with n genes on k chromosomes into another genome ψ of the same size but with h chromosomes, in polynomial time. As described in [4], this construction begins by combining the signed order representations of all the chromosomes in the two genomes. The following procedure, make-bicoloured, produces a bicoloured graph on $2n + 2k$

vertices that decomposes uniquely into a set of alternating-coloured cycles and set of $h + k$ alternating-colour paths. First, each gene or marker x in χ determines two vertices, x_t and x_h, to which two additional dummy vertices e_{i_1} and e_{i_2} are added to the ends of each chromosome χ_i. One colour edge, say red, is determined by the adjacencies in χ. If x is the left-hand neighbour of y in χ, and both have positive polarity, then x_h is connected by a red edge to y_t. If they both have negative polarity, it is x_t that is joined to y_h. If x is positive and y negative, or x is negative and y positive, x_h is joined to y_h, or x_t is joined to y_t, respectively. If x is the first gene in χ_i, then e_{i_1} is joined to x_t or x_h depending on whether x has positive or negative polarity, respectively. If x is the last gene, then e_{i_2} is joined to x_t or x_h depending on whether x is negative or positive.

Black edges are added according to the same rules, based on the adjacencies in genome ψ, though no dummy vertices are added in this genome.

It can be seen that each vertex is incident to exactly one red edge and one black edge, except for the dummy vertices in χ, which are each incident to only a red edge, plus the two (non-dummy) vertices at the ends of each chromosome in ψ, which are also each incident only to a red edge. The bicoloured graph decomposes uniquely into a number of alternating cycles plus $h + k$ alternating paths terminating in either the dummy vertices of χ or the end vertices of ψ, or one of each. Suppose the number of these paths that terminate in at least one dummy vertex (good paths) is $j \leq h + k$. If the number of cycles is c, then the minimum number of reversals r and translocations t necessary to convert χ into ψ is given by the Hannenhalli- Pevzner equation:

$$r + t = n - j - c + \theta \tag{4}$$

where θ is a correction term that is usually zero for simulated or empirical data. For simplicity of exposition, we ignore this correction here, though an eventual full-scale program will incorporate it with little or no computational cost.

5.2 Generalization to Partial Orders

The routine **make_bicoloured** can also be applied to the set of edges in the DGs for two partially ordered genomes. In the resulting graph, each of the DAG edges and both of the edges connecting each of the unordered pairs in the DG for each chromosome represent potential adjacencies in our eventual linearization of a genome. The n genes or markers and $2k$ dummies determine $2n + 2k$ vertices and the potential adjacencies determine the red and black edges, based on the polarity of the genes or markers. Where the construction for the totally ordered genomes contains exactly $n + k$ red edges and $n - h$ black edges, in our construction in the presence of uncertainty there are more potential edges of each colour, but only $2n + k - h$ can be chosen in our construction of the cycle graph, which equivalent to the simultaneous linearization by topological sorting of each chromosome in each genome. It is this problem of selecting the right subset of edges that makes the problem difficult (and, we conjecture, NP-hard.

The choice of certain edges generally excludes the choice of certain other ones. This is not just a question of avoiding multiple edges of the same colour incident to a single vertex. There are more subtle conflicts particularly involving the non-DAG edges, as illustrated in Figure 3. Our approach to this problem is a depth-first branch and bound search, **find_cycle_decomp**, in the environment of $h + k$ continually updated partial orders, one for each chromosome in each genome. The strategy is to build cycles and paths one at a time.

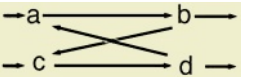


Fig. 3. If ab and cd are DAG edges, then the two non-DAG edges da and bc are mutually exclusive, since using them both leads to the wrong order for a and b.

Initially all edges in the DG for each chromosome are "eligible" and all vertices are"unused". We choose any initiating vertex u in the bicoloured graph and an edge ϵ connecting it to another vertex v. All remaining edges of the same colour incident to either u or v then become ineligible. For certain choices of ϵ, the partial order associated with that chromosome must be updated through the addition of the order relation represented by ϵ, plus all others involving one vertex ordered before u and one ordered after v.

At each successive stage of the search we add an eligible edge ϵ that does not conflict with the current partial order, incident to the most recently included vertex u to extend the current cycle or path to some as yet unused vertex v or, preferably, to close a cycle or complete a cycle or path. A complication is that when the construction reaches a potential end vertex in a chromosome of ψ, it is not always clearly the termination of a path since the DAG may contain several competing end vertices. (This is not a problem with the chromosomes in χ because the dummies e_{i_1} and e_{i_2} are always at the ends of the chromosomes.) In this case, the choice of path termination or not becomes one of the branches to explore in the branch and bound.

When an edge ϵ is added, the partial order for the chromosome containing ϵ is updated, if necessary, including whenever ϵ is not a DAG edge. If ϵ is in the DAG, no update is necessary (since the initial partial orders for the branch and bound are determined by the DAGs) unless u or v is incident to more than one eligible path of the same colour as ϵ, in which case additional order is imposed by the choice of ϵ. All remaining edges of the same colour incident to u or v are made ineligible and u is now "used".

When a cycle is completed, the initiating vertex also becomes "used". When a path is complete, both the initiating and terminating vertices become "used". Any unused vertex can then be chosen as to initiate a new cycle or path. (In our implementation, we increase the efficiency of the search by choosing a dummy or an end vertex whenever possible, including the very first choice of u.)

The search is bounded by using the fact that a cycle has at least two edges, and that a complete solution, representing some linearization, optimal or not,

always has $2n + k - h$ edges. Suppose the current best solution has c^* cycles and (necessarily) $h + k$ paths of which j^* are "good" as in (4). Suppose further the construction now underway is at a point where there are c' cycles and l paths, with j' good ones, and this has used m edges. This means there are only $2n + k - h - m$ edges left to chose of which at least $h + k - l$ must be in paths. Then the final number of cycles when the current construction is terminated will be no more than $c' + (2n + k - h - m - h - k + l)/2 = c' + n - h - (m - l)/2$. The final number of "good" paths will be no more than $j' + h + k - l$. So if

$$\begin{aligned} c' + n - h - (m - l)/2 + j' + h + k - l &= c' + j' + n + k - (m + l)/2 \\ &< c^* + j^*, \end{aligned} \tag{5}$$

this branch of the search is abandoned and backtracking begins.

Backtracking is also invoked if no cycles or paths can be made up of the unused vertices. During backtracking, when an edge is removed, so are the extra partial order relations it induced. The "eligible" and "unused" status it annulled are restored. An initial value of c^* can be found using any linearizations of the chromosomal DAGs of the two genomes or simply by running the depth-first algorithm until a first complete decomposition of the bicoloured graph is found.

6 Summary of the Analysis

The steps in our analysis, starting from several sets of incomplete chromosomal orders for each of two genomes, and outputting two genomes with totally ordered chromosomes, as well as a minimum number of reversals and translocations necessary to convert one to the other, are as follows.

Input: A number of incomplete maps for each genome
Remove: Genes or markers that do not appear in at least one map for each genome
For each chromosome in each map,
    make_po
For each genome,
    resolve
For each chromosome,
    combine_po
    dagger
    dgger
make_bicoloured
find_cycle_decomp
Output: Optimal cycles, paths and linearizations

The major time and space costs of our method are of course due to the branch and bound procedure in **find_cycle_decomp**. The number of potential edges to be considered for inclusion in the decomposition can grow as $O(n^2S^2)$, where S is the maximum number of parallel paths through the DAGS, but the depth of our search tree remains $O(n)$. The costs at each step are dominated by the necessity

of checking and updating a partial order matrix of size $O(n^2/h^2)$, assuming $h = k$, and all chromosomes are about the same size.

7 Analyzing the Simulated Incomplete Data

We submitted the data in (2) to our analysis. In (6) we compare the results to the original genomes in (2).

```
True genomes:
      {1  2   3  4  5  6  7,                  {-17 -3 -2 -1  18 -5 -4 -16,
     = 8  9  10 11 12 13 14 15,             =   8 -13 -12 -11 -22 -21,
      16 17 18 19 20,                           -7 -20  10 -24 -23  25,
      21 22 23 24 25}                          14  15,
                                                9 -19   6}
Reconstructed genomes:
      {1  3   2  5  4  6  7,                  {-17 -2 -3  -1  18 -4  -5 -16,
     = 8  9 10 11 12 13 14 15,              =   8 -13 -12 -11 -21 -22,
       16 17 18 19 20,                          -7 -20  10 -24 -23  25,
       22 21 23 24 25}                         14  15,
                                                9 -19   6}
```
(6)

The only reconstruction errors appear to be the reverse order of genes 2 and 3, 4 and 5, and 21 and 22 in both reconstructed genomes. An inspection of the simulated incomplete data in (3), however, shows that no information is present in any of the data sets for either genome that bears on the ordering within any of these pairs.

In addition, the reconstructed linearized chromosomes of χ in (6) require 4 translocations, 1 fission and four inversions to be transformed into the reconstructed ψ, exactly how ψ was originally obtained from χ in (2).

8 Performance and Future Work

The experimental version of our program can handle moderate size maps. Tests on simulated 6-chromosome maps with 100 genes, of which 10 % were missing, and 20% unresolved from each of three datasets for both of the two genomes being compared ($d(\chi, \psi) = 20$), executed in less than a second on a Macintosh G4. With 20% missing and 40% unresolved, an analysis usually required about 3 minutes. Increasing uncertainty beyond this quickly led to run times of several hours.

The current implementation is fairly straightforward, and there are a number of promising possibilities for increasing efficiency. Since many comparative maps have only a few hundred genes our method seems quite practical.

The one-sided version of our problem, where the chromosomes of one of the genomes being compared are totally ordered is of great interest. If one genome is known in great detail, we can then resolve many of the uncertainties in less densely mapped species, despite somewhat rearranged genomes, using our technique.

Acknowledgements

Research supported in part by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC). DS holds the Canada Research Chair in Mathematical Genomics and is a Fellow of the Evolutionary Biology Program of the Canadian Institute for Advanced Research.

References

1. Bader, D. A., Bernard M.E. Moret, B. M. E. and Yan, M. 2001. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. Journal of Computational Biology 8, 483–91.
2. Hannenhalli, S. and Pevzner, P.A. 1995. Transforming men into mice (polynomial algorithm for genomic distance problem. Proceedings of the IEEE 36th Annual Symposium on Foundations of Computer Science. 581–92.
3. Hannenhalli, S. and Pevzner, P.A. 1999. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). Journal of the ACM 48, 1–27.
4. Tesler, G. 2002. Efficient algorithms for multichromosomal genome rearrangements. Journal of Computer and System Sciences 65, 587–609.
5. Zheng, C., Lenert, A. and Sankoff, D. 2005. Reversal distance for partially ordered genomes. Bioinformatics 21, supplementary issue, Proceedings of ISMB 2005.

Quartet-Based Phylogeny Reconstruction from Gene Orders

Tao Liu[1], Jijun Tang[2], and Bernard M.E. Moret[1,⋆]

[1] Department of Computer Science, U. of New Mexico, Albuquerque, NM 87131
moret@cs.unm.edu
[2] Department of Computer Science & Engineering, U. of South Carolina, Columbia, SC 29208

Abstract. Phylogenetic reconstruction from gene-rearrangement data is attracting increasing attention from biologists and computer scientists. Methods used in reconstruction include distance-based methods, parsimony methods using sequence encodings, and direct optimization. The latter, pioneered by Sankoff and extended by us with the software suite GRAPPA, is the most accurate approach; however, its exhaustive approach means that it can be applied only to small datasets of fewer than 15 taxa. While we have successfully scaled it up to 1,000 genomes by integrating it with a disk-covering method (DCM-GRAPPA), the recursive decomposition may need many levels of recursion to handle datasets with 1,000 or more genomes. We thus investigated quartet-based approaches, which directly decompose the datasets into subsets of four taxa each; such approaches have been well studied for sequence data, but not for gene-rearrangement data. We give an optimization algorithm for the NP-hard problem of computing optimal trees for each quartet, present a variation of the dyadic method (using heuristics to choose suitable short quartets), and use both in simulation studies. We find that our quartet-based method can handle more genomes than the base version of GRAPPA, thus enabling us to reduce the number of levels of recursion in DCM-GRAPPA, but is more sensitive to the rate of evolution, with error rates rapidly increasing when saturation is approached.

1 Introduction

Modern techniques can yield the ordering and strandedness of genes for genomes; each chromosome can then be represented by an ordering of signed genes, where the sign indicates the strand. Rearrangement of genes under inversion, transposition, and other operations such as duplications, deletions and insertions, is an important evolutionary mechanism [7]. Reconstructing phylogenies from gene-order data has been studied intensely since the pioneering papers of Sankoff [2, 23]. Because they capture the complete genome, gene-order data do not suffer from the gene tree vs. species tree problem; and because rearrangements of genes are *rare genomic events* [19], gene-order data enable the reconstruction of evolutionary events far back in time. Many biologists have embraced this new source of data in their phylogenetic work [7, 17, 18], while computer scientists are slowly solving the difficult problems posed by the manipulations of these gene orders [16]. Studies conducted by our group [15, 26, 29–31] confirm that gene-order data support very accurate reconstructions.

⋆ Contact author

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 63–73, 2005.

The main software package for analyzing gene-order data is GRAPPA, based on the BPAnalysis software of Sankoff and Blanchette [23]. GRAPPA achieved a billion-fold speed-up over BPAnalysis [15]; it can analyze datasets of 13 genomes in 20 minutes on a laptop. Extensive testing has shown that the trees returned by GRAPPA are better than those returned by other methods based on gene orders, such as distance-based methods and parsimony based on encodings [6, 31]. However, since GRAPPA examines every possible tree, it can only handle small datasets – a 17-taxon analysis would take a month on today's most powerful computers. We integrated GRAPPA with DCM, a divide-and-conquer approach pioneered by Warnow [11], to produce DCM-GRAPPA [29], which can analyze datasets of up to 1,000 taxa without loss of accuracy.

DCM-GRAPPA works in three steps: it first decomposes the dataset into overlapping subproblems (disks), then runs GRAPPA on the subproblems, and finally uses a specialized supertree method [20] to build a tree for the original dataset from the trees returned by GRAPPA for the subproblems. Because the decomposition technique of DCM can still produce subproblems too large for GRAPPA to handle, we call DCM recursively until each subproblem size falls below a given threshold. Because the threshold is small (14), large problems require many levels of recursive decomposition, which is time-consuming and also risks propagating and amplifying errors in the assembly of the subtrees. On 1,000 genomes, DCM-GRAPPA needs 6–7 levels of recursion if limited to disks of at most 14 genomes, but only 2–3 levels if allowed disks of up to 20 genomes.

One can also decompose the set of taxa into the smallest possible subsets for which meaningful answers exist, namely *quartets*, sets of four taxa. (Sets of two or three taxa can produce only one tree, but a quartet can give rise to three distinct unrooted trees.) While there are many such quartets, their tiny size should make them easy to compute. If every quartet tree is computed correctly from noiseless data, then there is a single tree compatible with all $\binom{n}{4}$ resolved quartets and that tree is the true tree [4]; in practice, of course, many of the resolved quartets are in error and no single tree is compatible with all resolved quartets. With sequence data, biologists have long used the heuristic method known as quartet-puzzling [25], while computer scientists have developed several theoretical methods, such as quartet cleaning [1, 3, 12] – see [24] for a review and experimental comparison of these methods. In the case of gene orders, however, optimally resolving a quartet is NP-hard – it includes finding the median of three genomes, a known NP-hard problem [5], as a special case.

We present algorithmic and experimental results that lead to a reconstruction method from gene-order data which overcomes some of the problems associated with quartet methods. After some background review and definitions, we describe in Section 3 our exact method to compute optimal quartet trees under breakpoint distances; in Section 4.1 we present our experimental studies to find the best methods to resolve the quartets, as well as our use of the dyadic inference rule (see [10]) to obtain a sufficient set of quartets from a selected subset of short quartets (inspired from the short-quartet method [8, 32]); in Section 5, we summarize our experiments on simulated and biological datasets, the results of which suggest that our method can produce accurate topologies (much more accurate than neighbor-joining) for datasets of up to 25 taxa within reasonable time, provided that the genomes are large enough to avoid saturation.

2 Definitions and Notation

A quartet is a quadruple of taxa; a quartet tree is an unrooted binary tree for such a quadruple. Given a quartet $\{a, b, c, d\}$, we say that a quartet tree on this set is *unresolved* if it is a star (four edges, each touching a leaf) and denote it by $(abcd)$. If the quartet tree has an internal edge separating two pairs of leaves we say that it is *resolved* and, if the pairs are a, b and c, d, denote it by $ab \mid cd$. The four possible quartet trees induced by a quartet are depicted in Figure 1. We will use quartet in lieu of quartet tree or resolved quartet when the sense is clear.

Quartet tree $ab \mid cd$ *agrees* with tree T if all four of its taxa are leaves of T and the path from a to b in T does not intersect with the path from c to d in T. Equivalently, $ab \mid cd$ agrees with a tree if the subtree induced in T by the four-taxon subset $\{a, b, c, d\}$ is the quartet tree itself. Quartet $ab \mid cd$ is *in error* with respect to the tree T if it does not agree with T. If Q_T denotes the set of all quartets that agree with T, then T is uniquely characterized by Q_T; moreover, T can be reconstructed in polynomial time from Q_T [8]. (Of course, the set Q of $\binom{n}{4}$ quartets that we can construct is only an approximation of Q_T.) In Figure 2, quartet $ac \mid bd$ would be in error, since it does not appear in Q_T.

Quartet-based methods operate in two phases. First, they construct a set Q of resolved quartets – usually by determining the preferred tree for each of the $\binom{n}{4}$ quartets. Because each dataset has size 4, any phylogenetic method can be used to estimate the quartet tree, including maximum likelihood and maximum parsimony (see [27]), neighbor-joining [21], the relaxed four-point method [9], and the ordinal quartet method [13]. In the second phase, the resolved quartets are used to build a single tree on the full set of taxa. This second phase is simple when all quartets are compatible, but a challenge when some of the quartets conflict with others – a common occurrence when insufficient data are present [24].

3 Inferring Quartet Topologies

We can identify maximally parsimonious quartet trees by examining each of the three possible trees, assigning gene orders to the two internal nodes so as to minimize the score (the sum of the lengths of the five edges) of each tree, and returning the tree with the lowest score. However, identifying such gene orders is NP-hard even for just one internal node and the simplest distance measures [5].

An easy approach is to use `GRAPPA` to construct a tree for each quartet; although the result need not be optimal, our experiments, as well as earlier ones [14], show that optimality is reached in most realistic cases. If we limit ourselves to breakpoint distances,

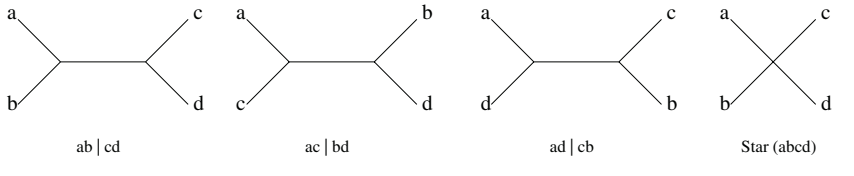


Fig. 1. The four possible quartet trees for quartet $\{a, b, c, d\}$.

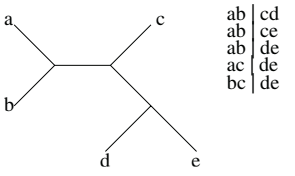


Fig. 2. An evolutionary tree T and its set Q_T of induced quartet trees.

we can solve the NP-hard optimization problem directly, using variants of the reduction to TSP devised by Sankoff and Blanchette [22]; we devised two such variations, which we call `Qtsp` and `Qedge`.

Sankoff's reduction to TSP can be summarized as follows. Given three genomes with n genes, we build the complete graph K_{2n} on the $2n$ vertices g_1, $-g_1$, g_2, $-g_2$, ..., g_n, $-g_n$. Edge (g, h) in this graph is assigned a weight as follows: if we have $g = -h$, then we set the weight to a large negative value to ensure that (g, h) is part of any solution, otherwise we set it to $3 - \text{adj}(g, h)$, where $\text{adj}(g, h)$ is the number of times that $-g$ and h are adjacent in the given genomes. If $s = s_1, -s_1, s_2, -s_2, \cdots s_n, -s_n$ is the solution to the TSP, then the median of the given genomes is $g = s_1, s_2, \cdots, s_n$.

This result applies to one unknown genome, but we need to identify two such for quartets. Our first algorithm views the two as forming a pair and remaps the problem into a universe where pairs form the unit of computation and where a single tour (of pairs) defines both genomes; our second algorithm retains the original formulation, but looks for a pair of tours.

3.1 The `Qtsp` Method

We look for one permutation of size n, each entry of which consists of a pair of genes (g_i, g_j). Let the two desired internal genomes be e and f, where genome e is connected to genomes a, b, and f, and genome f is connected to genomes c, d, and e. Build a complete graph, here on $(2n)^2$ vertices, each a pair of signed genomes. For each edge $\{(g_{i1}, g_{i2}), (g_{j1}, g_{j2})\}$ in this graph, we set the weight of the edge as follows: if we have $g_{i1} = g_{j1}$ and $g_{i1} = g_{j1}$, then we set the weight to a large negative value to ensure that this edge is part of any solution, otherwise, we set it to $4 - u_1(g_{i1}, g_{j1}) - u_2(g_{i2}, g_{j2})$, where $u_1(g_{i1}, g_{j1})$ is the number of times $-g_{i1}$ and g_{j1} are adjacent in the genomes a and b and $u_2(g_{i2}, g_{j2})$ is the number of times $-g_{i2}$ and g_{j2} are adjacent in the genomes c and d.

Proposition 1. *If* $s = (s_{11}, s_{12}), (-s_{11}, -s_{12}), \cdots, (s_{n1}, s_{n2}), (-s_{n1}, -s_{n2})$ *is the solution to the TSP on G, then the optimal internal genomes are* $e = s_{11}, s_{21}, \cdots, s_{n1}$ *and* $f = s_{12}, s_{22}, \cdots, s_{n2}$.

After transforming the problem, we can use the efficient TSP routine of `GRAPPA` to search for the optimal solution (after modifying it to introduce one more bucket of costs, since now the edge costs of interest are 1, 2, and 3, not just 1 and 2). The problem is that the instance thus created is of size quadratic in the number of genes; combined with

the extra bucket of costs, the large instance size makes it difficult to obtain solutions to sizeable instances.

3.2 The `Qedge` Method

This method uses the original TSP formulation, of size linear in the number of genes, but seeks simultaneously to optimize tours in two separate graphs. Let a, b, c, d, e, and f be as before. We set up two complete graphs on $2n$ vertices each – one for e and one for f. For each edge $\{g, h\}$, we set its weight as follows: if we have $g = -h$, we set the weight to a large negative value to force its inclusion; otherwise, we set it to $2 - u(gh)$, where $u(gh)$ is the number of times $-g$ and h are adjacent in the genomes a and b (for an edge in the first graph) or in the genomes c and d (for an edge in the second graph). Now, when adding an edge to the two tours under construction, we can either pick different edges from the two graphs, each with the minimum weight in its own graph, and add one breakpoint between e and f to the total cost; or pick the same edge in both graphs, even if not locally optimal, thereby saving a breakpoint between e and f; our algorithm computes the cost of each choice and picks the choice of lower cost.

3.3 Experimental Results for `Qtsp` and `Qedge`

We ran tests for these two methods and `GRAPPA` on quartets of 10, 20, 30, 40, and 50 genes under evolutionary rates (the expected numbers of events per edge of the model tree) of $r = 1$, 2, 3, 4, and 5. The running times of `Qtsp` and `Qedge` are shown in Figure 3. As expected, `Qedge` runs much faster than `Qtsp`; however, its running time depends strongly on the quartet score, so that it may prove unusable in the reconstruction of large trees, where many of the quartets will have very large scores. Both `Qtsp` and `Qedge` reconstructed the same optimal quartet trees (except for ties), which were the same as the model tree in 97% of cases (though their scores were usually lower than the model tree scores); most of the 3% came from the 10-gene case, which gets quickly saturated for evolutionary rates above $r = 3$. `GRAPPA` did almost as well: 96% of the quartets it returned matched the model tree – and it returned answers in a few

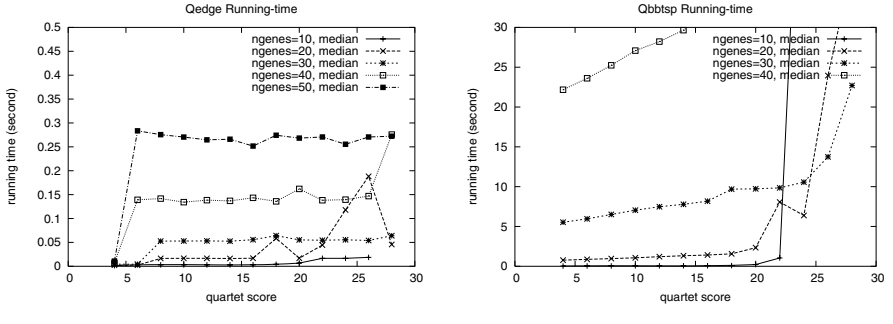


Fig. 3. Running times of `Qtsp` (left) and `Qedge` (right) as a function of quartet score.

microseconds. Since reconstructing phylogenies from quartets requires the computation of a large set of quartets, the running time is critical. Therefore, based on our results, we chose GRAPPA as the method to resolve the quartets.

4 Phylogenetic Reconstruction from Quartets

The computational challenge of quartet recombination (building a single tree from the collection of quartet trees) is how to deal with quartet errors. Most optimization problems related to tree reconstruction from quartets are NP-hard, such as the Maximum Quartet Compatibility problem [12], which seeks a tree T for a given set of quartet Q such that $|Q_T \cap Q|$ is maximized. Various methods have been designed to handle quartet errors. The dyadic-closure method simply issues an error message and quits [10]. The $Q*$ method seeks the maximum resolved tree T' that obeys $Q(T') \subseteq Q$, a conservative method that generally produces many polytomies [4]. Quartet-cleaning methods establish a bound on the number of quartet errors around each reconstructed tree edge [1, 3, 12]. None of these methods produces satisfactory results on sequence data [24]. It is theoretically possible to produce the true tree by selecting a subset of short quartets and adding to these further quartets derived according to an inference rule [8], but this result assumes perfect data and gives no simple method by which to select the subset of quartets. We thus set out to design a selection rule and investigate its performance, using the dyadic inference rules [10]:

1. If $ab \mid cd$ is a valid quartet, so are $ba \mid cd$ and $cd \mid ab$.
2. If $ab \mid cd$ and $ac \mid de$ are valid quartets, so are $ab \mid ce$, $ab \mid de$, and $bc \mid de$.
3. If $ab \mid cd$ and $ab \mid ce$ are valid quartets, so is $ab \mid de$.

4.1 Selecting a Subset of Quartets

Figure 4 shows two possible resolutions for quartet $\{a, b, c, d\}$. In the first topology, the two pairs of genomes $\{a, b\}$ and $\{c, d\}$ are far apart from each other, but in the second topology the two pairs $\{a, c\}$ and $\{b, d\}$ are quite close: the first topology is more likely to be correct, an observation supported by the relaxed four-point method [9]:
Compute pairwise distances among a, b, c, and d; return $ab \mid cd$ *if we have* $d_{ab} + d_{cd} < min(d_{ac} + d_{bd}, d_{ad} + d_{bc})$*, but return a star if all sums are equal.* Since computing all $\binom{n}{4}$ quartets takes too long, we can use this relaxed four-point method to choose quartets and reduce the overall running time. After resolving the quartets, we can assign a weight to each resolved quartet to measure our confidence in that quartet: for example, we can use the inversion distance between the two internal nodes.

Fig. 4. Two quartet trees; the left has a higher probability of correctness.

4.2 Fixing Quartet Errors

Although GRAPPA is reliable and although we can pick only quartets of larger weight, quartet errors still arise, especially when we are forced to select some quartets of low weight in order to resolve every internal tree edge. We propose a simple new method, quite distinct from quartet cleaning, to handle errors. Since the quartets are weighted and since we place more trust in quartets of higher weight, we examine the source of quartet errors and, whenever two quartets are incompatible, we remove the one with lower weight.

Starting from a large initial set of resolved quartets only returns us to a version of the NP-hard problem Maximum Quartet Compatibility. Instead, we proceed incrementally. We select a high weight threshold and only retain quartets (computed on the fly with GRAPPA) with weights above that threshold; if we find quartet errors, we remove the incompatible quartets of lower weight. We then apply the dyadic inference rules to augment our collection of compatible quartets. Finally, if the resulting set of quartets fully resolves the tree, we are done (a method like $Q*$ will recover the tree), otherwise we lower the threshold and add to our set the newly eligible quartets. By controlling the decrease in the weight threshold, we can control the tradeoff between running time and quality.

Since we do not know the weight of a quartet until we resolve it, but want to avoid resolving useless quartets, we need a fast method to select quartets to resolve. Given quartet $q = \{a, b, c, d\}$, define the *width* of q as
$q_w = max(d_{ab} + d_{cd}, d_{ac} + d_{bd}, d_{ad} + d_{bc}) - min(d_{ab} + d_{cd}, d_{ac} + d_{bd}, d_{ad} + d_{bc})$
As q_w increases, the two pairs of genomes move further apart and the weight increases: hence we can decide which quartets to resolve by comparing their width with the weight threshold. Even if the threshold is lowered to zero, the set of compatible quartets may remain inadequate to resolve the tree – in which case we have no choice but to leave these unresolved polytomies in the output.

5 Experimental Results

If the true tree has an edge defining a bipartition with no equivalent in the reconstructed tree, that edge is a *false negative (FN)*; conversely, if the reconstructed tree has an edge with no equivalent in the true tree, that edge is a *false positive (FP)*. FP edges are more problematic than FN edges.

We generated model tree topologies from the uniform distribution on binary trees, each with 12, 16 and 20 leaves respectively. On each tree, we evolved signed permutations of 40, 60 and 80 genes, using evolutionary rates (the expected numbers of events along a tree edge) of 2, 4, 6. For each combination of parameters, we generated 20 trees; the final results are averaged on the 20 datasets. We computed quartets using GRAPPA and built the resulting tree using our algorithms for selecting quartets of high weight, eliminating conflicting quartets, and expanding the set with the dyadic rules. Figure 5 shows FP and FN rates for datasets with 80 genes. Our method did well, but saturation (high evolutionary rates leading to ill-defined estimates of distances) causes a small increase in the error rate. This observation is confirmed by our results on datasets with 40 genes, where saturation occurs much sooner and the results are unacceptable, as seen in Figure 6. It can also be seen in Figure 5 that, with very low evolutionary rates, many

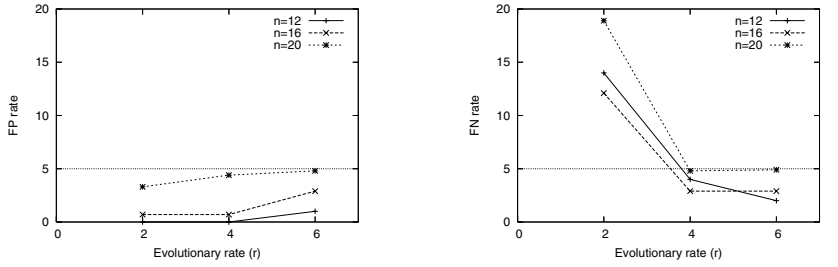


Fig. 5. Performance of our method on genomes of 80 genes.

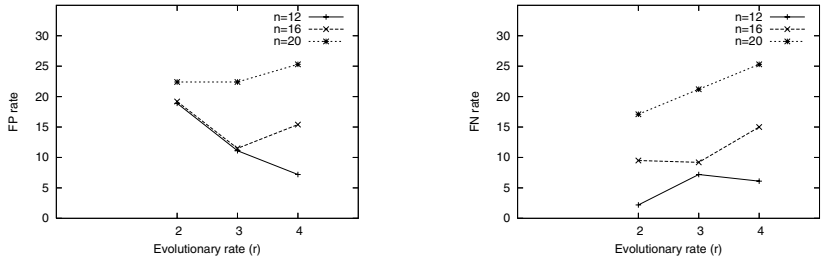


Fig. 6. Performance of our method on genomes of 40 genes.

quartets cannot be satisfactorily resolved (we get equalities and thus star topologies), leading to poor resolution and many false negatives.

Our tests verified that a small subset of quartets suffices to infer the complete set of quartets. For datasets with 12 genomes, only 75 quartets (15% of the total) are needed; with 20 genomes, only 270 quartets (6%) are needed. Our selection rule worked well: of the quartets selected, fewer than 2% overall were found to be incompatible. Interestingly, the set of resolved quartets produced by our method produced very accurate reconstructions, while the set produced directly by the relaxed four-point method gave very poor results.

We compared the results obtained by our method with those obtained by simply running the (very fast) neighbor-joining (NJ) method on breakpoint and inversion distance matrices (computed by GRAPPA) for each dataset. For 80-gene genomes, the Robinson-Foulds rate (the average of FP and FN rates) for NJ varied from 20% (for $r = 2$) down to 2–5% (for $r = 6$, with lower rates for 12 genomes and larger rates for 20 genomes), compared to a maximum of 10% (for $r = 2$) down to 1.5–4.5% (for $r = 6$) for our method. For 40-gene genomes, as we observed, our method suffers from saturation effects with 16 or 20 genomes, where its error rate roughly matches that of NJ (10-20%); for 12 genomes, where saturation is less of a problem, our method again easily surpasses NJ, with a median error rate of 8.5% compared to NJ's rate of 14%.

6 Conclusions

We have presented a quartet-based phylogeny reconstruction method for gene-order data and reported its performance on simulated datasets. Our method produces accurate

topologies for trees with up to 25 leaves in reasonable time when the datasets do not exhibit significant saturation. The results we have obtained promise well, especially because we have many possible avenues of improvement. For instance, we have recently developed a linear-programming method that can accurately estimate the edge lengths of fairly small trees [28]; by using this method to estimate the length of quartet edges, we can further improve our quartet selection, in terms of both speed and accuracy. Such improvement should also enable us to handle datasets with larger pairwise distances. We designed this method to extend the range of base methods that can be used in conjunction with a disk-covering method: thus the limitation to sets of 20–30 taxa is not an issue, but in fact a potentially significant gain over the direct use of `GRAPPA` as a base method, since this last is limited to 12–15 taxa.

Acknowledgments

This work is supported by the US National Science Foundation under grants EF 03-31654, IIS 01-13095, IIS 01-21377, and DEB 01-20709, by the US National Institutes of Health under grant 2R01GM056120-05A1 (through a subcontract to the U. of Arizona), and by the Dept. of Computer Science and Engineering at the U. of South Carolina.

References

1. V. Berry, T. Jiang, P. Kearney, M. Li, and T. Wareham. Quartet cleaning: improved algorithms and simulations. In *Proc. Europ. Symp. Algs. (ESA99)*, volume 1643 of *Lecture Notes in Computer Science*, pp. 313–324. Springer Verlag, 1999.
2. M. Blanchette, G. Bourque, and D. Sankoff. Breakpoint phylogenies. In S. Miyano and T. Takagi, editors, *Genome Informatics 1997*, pp. 25–34. Univ. Academy Press, 1997.
3. D. Bryant, V. Berry, T. Jiang, P. Kearney, M. Li, T. Wareham, and H. Zhang. A practical algorithm for recovering the best supported edges of an evolutionary tree. In *Proc. 11th Ann. ACM/SIAM Symp. Discrete Algs. (SODA'00)*, pp. 287–296. ACM Press, New York, 2000.
4. P. Buneman. *The recovery of trees from measures of dissimilarity*. Edinburgh University Press, 1971.
5. A. Caprara. Formulations and hardness of multiple sorting by reversals. In *Proc. 3rd Ann. Int'l Conf. Comput. Mol. Biol. (RECOMB'99)*, pp. 84–93. ACM Press, New York, 1999.
6. M.E. Cosner, R.K. Jansen, B.M.E. Moret, L.A. Raubeson, L. Wang, T. Warnow, and S.K. Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics*, pp. 99–122. Kluwer Academic Publishers, 2000.
7. S.R. Downie and J.D. Palmer. Use of chloroplast DNA rearrangements in reconstructing plant phylogeny. In P. Soltis, D. Soltis, and J.J. Doyle, editors, *Plant Molecular Systematics*, pp. 14–35. Chapman and Hall, 1992.
8. P. Erdős, M. A. Steel, L. A. Székely, and T. Warnow. A few logs suffice to build (almost) all trees I. *Random Structs. and Algs.*, 14:153–184, 1997.
9. P. L. Erdős, M. A. Steel, L. A. Székely, and T. Warnow. Constructing big trees from short sequences. In *Proc. 24th Int'l Colloq. on Automata, Languages, and Programming (ICALP97)*, volume 1256 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

10. P. L. Erdős, M. A. Steel, L. A. Székely, and T. Warnow. Local quartet splits of a binary tree infer all quartet splits via one dyadic inference rule. *Computers and Artif. Intell.*, 16(2):217–227, 1997.
11. D. Huson, S. Nettles, and T. Warnow. Disk-covering, a fast converging method for phylogenetic tree reconstruction. *J. Comput. Biol.*, 6(3):369–386, 1999.
12. T. Jiang, P.E. Kearney, and M. Li. A polynomial-time approximation scheme for inferring evolutionary trees from quartet topologies and its application. *SIAM J. Computing*, 30(6):1942–1961, 2001.
13. P.E. Kearney. The ordinal quartet method. In *Proc. 2nd Ann. Int'l Conf. Comput. Mol. Biol. (RECOMB'98)*, pp. 125–134. ACM Press, New York, 1998.
14. B.M.E. Moret, A.C. Siepel, J. Tang, and T. Liu. Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. In *Proc. 2nd Int'l Workshop Algs. in Bioinformatics (WABI'02)*, volume 2452 of *Lecture Notes in Computer Science*, pp. 521–536. Springer Verlag, 2002.
15. B.M.E. Moret, J. Tang, L.-S. Wang, and T. Warnow. Steps toward accurate reconstructions of phylogenies from gene-order data. *J. Comput. Syst. Sci.*, 65(3):508–525, 2002.
16. B.M.E. Moret, J. Tang, and T. Warnow. Reconstructing phylogenies from gene-content and gene-order data. In O. Gascuel, editor, *Mathematics of Evolution and Phylogeny*, pp. 321–352. Oxford University Press, 2005.
17. J.D. Palmer. Chloroplast and mitochondrial genome evolution in land plants. In R. Herrmann, editor, *Cell Organelles*, pp. 99–133. Springer Verlag, 1992.
18. L.A. Raubeson and R.K. Jansen. Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. *Science*, 255:1697–1699, 1992.
19. A. Rokas and P.W.H. Holland. Rare genomic changes as a tool for phylogenetics. *Trends in Ecol. and Evol.*, 15:454–459, 2000.
20. U. Roshan, B.M.E. Moret, T. Warnow, and T.L. Williams. Performance of supertree methods on various dataset decompositions. In O.R.P. Bininda-Edmonds, editor, *Phylogenetic Supertrees: Combining information to reveal the Tree of Life*, pp. 301–328. Kluwer Academic Publishers, 2004.
21. N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 4:406–425, 1987.
22. D. Sankoff and M. Blanchette. The median problem for breakpoints in comparative genomics. In *Proc. 3rd Int'l Conf. Computing and Combinatorics (COCOON'97)*, volume 1276 of *Lecture Notes in Computer Science*, pp. 251–264. Springer Verlag, 1997.
23. D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *J. Comput. Biol.*, 5:555–570, 1998.
24. K. St. John, T. Warnow, B.M.E. Moret, and L. Vawter. Performance study of phylogenetic methods: (unweighted) quartet methods and neighbor-joining. *J. Algorithms*, 48(1):173–193, 2003. A preliminary version appeared in SODA'01, pp. 196–205.
25. K. Strimmer and A. von Haeseler. Quartet puzzling: A quartet maximum likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.*, 13:964–969, 1996.
26. K.M. Swenson, M. Marron, J.V. Earnest-DeYoung, and B.M.E. Moret. Approximating the true evolutionary distance between two genomes. In *Proc. 7th SIAM Workshop on Algorithm Engineering & Experiments (ALENEX'05)*. SIAM Press, Philadelphia, 2005.
27. D.L. Swofford, G.J. Olsen, P.J. Waddell, and D.M. Hillis. Phylogenetic inference. In D.M. Hillis, B.K. Mable, and C. Moritz, editors, *Molecular Systematics*, pp. 407–514. Sinauer Assoc., Sunderland, MA, 1996.
28. J. Tang and B.M.E. Moret. Linear programming for phylogenetic reconstruction based on gene rearrangements. In *Proc. 16th Ann. Symp. Combin. Pattern Matching (CPM'05)*, Lecture Notes in Computer Science, 2005.

29. J. Tang and B.M.E. Moret. Scaling up accurate phylogenetic reconstruction from gene-order data. In *Proc. 11th Int'l Conf. on Intelligent Systems for Mol. Biol. (ISMB'03)*, volume 19 of *Bioinformatics*, pp. i305–i312. Oxford U. Press, 2003.
30. J. Tang, B.M.E. Moret, L. Cui, and C.W. dePamphilis. Phylogenetic reconstruction from arbitrary gene-order data. In *Proc. 4th IEEE Symp. on Bioinformatics and Bioengineering BIBE'04*, pp. 592–599. IEEE Press, Piscataway, NJ, 2004.
31. L.-S. Wang, R.K. Jansen, B.M.E. Moret, L.A. Raubeson, and T. Warnow. Fast phylogenetic methods for genome rearrangement evolution: An empirical study. In *Proc. 7th Pacific Symp. on Biocomputing (PSB'02)*, pp. 524–535. World Scientific Pub., 2002.
32. T. Warnow, B.M.E. Moret, and K. St. John. Absolute convergence: true trees from short sequences. In *Proc. 12th Ann. ACM/SIAM Symp. Discrete Algs. (SODA'01)*, pp. 186–195. SIAM Press, 2001.

Algorithmic and Complexity Issues of Three Clustering Methods in Microarray Data Analysis (Extended Abstract)

Jinsong Tan[1], Kok Seng Chua[2], and Louxin Zhang[1,⋆]

[1] Department of Mathematics
National University of Singapore, Singapore 117543
{matzlx,mattjs}@nus.edu.sg
[2] The Inst. of High Performance Computing
Singapore 117528

Abstract. The complexity, approximation and algorithmic issues of several clustering problems are studied. These non-traditional clustering problems arise from recent studies in microarray data analysis. We prove the following results. (1) Two variants of the Order-Preserving Submatrix problem are NP-hard. There are polynomial-time algorithms for the Order-Preserving Submatrix Problem when the condition or gene sets are given. (2) The Smooth Subset problem cannot be approximable with ratio $0.5 + \delta$ for any constant $\delta > 0$ unless NP=P. (3) Inferring plaid model problem is NP-hard.

1 Introduction

Clustering analysis is a vital step in microarray experiment. It gives a readout of the distinct patterns of genes switched on or off in a cell and hence allows a researcher have a comprehensive snapshot of the cellular dynamics in a condition (such as tissues, environments) (e.g. [1, 9, 20]). The analysis is divided into two steps for revealing common patterns of gene expressions across different conditions. The first step is to arrange gene-expression values into a matrix, in which the rows represent genes, the columns represent conditions, and hence each entry is a measure of the expression strength of a gene in a condition. Based on this matrix, we may treat each gene as a vector (or point) in an n-dimensional metric space, where n is the number of the conditions. The genes are then clustered into groups by a method that measures the distances between their corresponding vectors. Clustering analysis can also group conditions that show similar patterns of genome-wide gene expression.

Traditional methods include k-means, self-organizing maps [20], and hierarchical clustering [9]. Many new methods have also been proposed for the following reasons. First, the traditional methods are best suited to determining

⋆ Corresponding Author. This work was partially supported by the grant BMRC01/1/21/19/140.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 74–83, 2005.

relationships among a small number of variables, rather than deriving expression patterns involving thousands of genes. Secondly, microarray experiments have relatively low sensitivity. When rare diseases are studied, there are not enough samples and hence gene expression data missing occurs inevitably. For instance, in the gene expression data with 4026 genes and 96 conditions used in [1], there are 47639 missing values, which is about 12% of the total values.

Here, we study the complexity and algorithmic issues of non-traditional clustering methods that are recently proposed in [4], [17], and [24]. Being motivated by the fact that a subset of genes co-express under some but not all conditions, the authors of these works focused on finding local expression patterns on a subset of genes and/or experimental conditions.

In [4], Ben-Dor *et al.* studied the problem of identifying order-preserving submatrices, in which all genes co-express in the same magnitude under the conditions. In this paper, we prove the NP-hardness of two versions of the order-preserving submatrix problem and presents a quadratic-time algorithm for two practical subcases of the problem. We also validate the proposed algorithms on real data sets.

Based on the so-called smooth score, Zhang and Zhu proposed a clustering method aiming for overcoming data errors such as data missing [21] and data inconsistency [6] in the stage of clustering analysis. The smooth score is not defined as a pairwise dissimilarity measure like Euclidean distance; instead, it measures the deviation of the expression level of a gene from the average expression level of all concerned genes under a condition (see Formula (1) for details). In the paper [24], they proposed efficient greedy algorithms for the Smooth Clustering problem: given a set of conditions, find a largest cluster of genes with its smooth score below a threshold under the given conditions. They also looked for a largest smooth 'bicluster', grouping genes and conditions simultaneously as proposed in [7]. Here, we study the approximation issue of a variant of the smooth clustering problem. The smooth clustering problem is similar to the tiling problem with rectangles [5].

Finally, in their paper [17], Lazzeroni and Owen introduced the so-called plaid model for exploratory analysis of microarray data in statistical approach. The plaid model seeks the decomposition of a gene expression matrix into submatrices with uniform entries. This is a very general statistical model. The decomposition methods related to it include singular value decomposition, semidiscrete decomposition [14], and non-negative matrix factorization [18]. Here, we show that inferring an optimal plaid model is NP-hard. This answers an open problem posed in [17].

For basic notations and knowledge on NP-hardness and approximation algorithms, the reader is referred to [2, 10, 13].

2 The Order-Preserving Submatrix Problem

In the rest of this paper, we use $A = (a_{ij})$ to denote a gene expression matrix with gene set X and condition set Y, in which a_{ij} denotes the expression value of the ith gene in the jth condition. We also use $|X|$ and $|Y|$ to denote the numbers of genes in X and conditions in Y respectively.

2.1 The Problem

Each gene (in the gene expression data $A = (a_{ij})$) induces an ordering of all the conditions in terms of its expression values. Obviously, two genes in rows i and j induce the same linear ordering if for any different $k, k' \in Y$, $a_{ik} \neq a_{ik'}$ and $a_{ik} - a_{ik'}$ has the same sign as $a_{jk} - a_{jk'}$.

An *order-preserving submatrix* of A corresponds to a subset X' of genes and a subset Y' of conditions such that, within conditions in Y', the expression levels of all the genes in X' have the same linear ordering. The problem of identifying a large order-preserving submatrix is formally defined as [3]:

Order-Preserving Submatrix (OPSM)
Instance: A gene expression matrix $A = (a_{ij})$ with gene set X and condition set Y.
Question: Find an order-preserving submatrix $A(I, J)$, $I \subseteq X$ and $J \subseteq Y$, that maximizes $\min\{|I|, |J|\}$.

2.2 NP-Completeness

We first prove the following variant of the OPSM problem is also NP-complete:
Order-Preserving Submatrix II
Instance: A gene expression matrix $A = (a_{ij})$ with gene set X and condition set Y, and an positive integer k
Question: Find an OPSM $A(I, J)$ of k entries, where $I \subseteq X$ and $J \subseteq Y$.

Theorem 1. *The Order-Preserving Submatrix II problem is NP-complete.*

Proof. (Sketch of the proof) The NP-completeness proof is through a reduction from the Maximum Edge Biclique problem:
Instance: A bipartite graph $G = (V, E)$, and positive integer $k \leq |E|$.
Question: Does G contain a complete bipartite subgraph with at least k edges. which is proved to be NP-complete in [19] recently. □

We then consider the following problem that is closely related to the OPSM problem. It arises from finding genetic features in cancer class discovery and prediction.
Maximum Differential Gene Subset
Instance: A gene expression matrix $A = (a_{ij})$ with gene set X and condition set Y, and two positive integers $k \leq |X|$ and $s \leq |Y|/2$.
Question: Are there a gene subset $X' \subseteq X$ and two disjoint condition subsets $Y', Y'' \subset Y$ such that $|X'| = k$, $|Y'| = |Y''| = s$ and such that for each $i \in X'$, $j' \in Y', j'' \in Y''$, $a_{ij'} < a_{ij''}$?

Theorem 2. *The Maximum Differential Gene Subset problem is NP-complete.*

Proof. It is proved through a reduction from the Balanced Complete Bipartite Subgraph (BCBS) problem. The details are omitted.

2.3 Efficient Algorithms for Special Cases

Since the OPSM problem is NP-hard, it is unlikely polynomial-time solvable. In this subsection, we present an efficient algorithm for two practical cases of this problem, which leads to a feasible approach for microarray data analysis.

Theorem 3. *For a given gene expression matrix $A = (a_{ij})$ with gene set X and condition set Y, the following two variants of the OPSM problem are linear-time and quadratic-time solvable, respectively.*

i) Given a subset $J \subseteq Y$, find a largest subset $I \subseteq X$ such that $A(I, J)$ is order-preserving.

ii) Given a subset $I \subseteq X$, find a largest subset $J \subseteq Y$ such that $A(I, J)$ is order-preserving.

Proof. i). The idea of the proof in this case is simple. Each gene induces a linear ordering on the condition subset J. Note that $A(I, J)$ is order-preserving if and only if all the genes in I induce same ordering. Hence, we can sort the orderings induced by all the genes in X and take the largest subset of genes that incur the same ordering. With radix sort [8], this algorithm can be implemented in $O(|X|)$ time since the size of J is fixed.

ii). It is less obvious how to find a largest subset $J \subseteq Y$, given $I \subseteq X$, such that $A(I, J)$ is order-preserving in polynomial time. Here, we reduce it to the problem of finding a longest path in acyclic graphs. Given a gene expression matrix $A = (a_{ij})$ with gene set X and condition set Y and a gene subset $I \subseteq X$, we define a directed graph $D_A = (V_A, E_A)$, where V_A contains $|Y|$ vertices each corresponding to a condition in Y, and there is an arc $(u, v) \in E_A$ from u to v if their corresponding conditions j_u and j_v satisfy that for any $i \in I$, $a_{ij_u} < a_{ij_v}$. Obviously, D_A is acyclic and can be constructed in quadratic time. Furthermore, for any condition subset J, $A(I, J)$ is order-preserving if and only if the vertices corresponding to conditions in J form a directed path in D_A. This implies that we only need to find a longest path in the directed graph D_A, which is solvable in linear time $O(|V_A| + |E_A|)$ (see for example [15]). The algorithm will be given in the full version of this manuscript. □

2.4 Validation Experimental Test

Most of the microarray data usually contain about 10 to 30 conditions. Given an integer $k \leq 20$, there are only about one million different k-condition subsets. Therefore, by applying Theorem 3 on each of such condition subset, we are able to find a k-condition subset $J \subseteq Y$ and $I \subseteq X$ such that $A(I, J)$ is order-preserving and $|I|$ is maximal. We implemented a program based on the first algorithm in Theorem 3 and validated it through a real microarray dataset.

On an input expression matrix A with gene set X and condition set Y, for each value $k \leq |Y|$, the program identifies all the largest order-preserving submatrices induced by some k-condition subset by enumerating all the k-condition subsets. Then, the statistical significance of each obtained submatrix is evaluated for acceptance.

We test the program on the breast tumor dataset reported in [12] on a Linux machine with 2.4G Pentium 4 CPU. The dataset consists of 3226 genes and 22 conditions. We compared our program to the original OPSM program in [4] and another program in [16]. We find that our algorithm found not only all the biologically meaningful clusters reported in [4] and [16], but also many more other statistically significant clusters. The testing details can be obtained from the authors.

3 The Smooth Clustering Problems

3.1 Definitions

Any subsets $I \subseteq X$ and $J \subseteq Y$ specify a submatrix $A(I, J)$. We associate it with a smooth score

$$s(I, J) = \max_{j \in J} (\max_{i \in I} |a_{ij} - \frac{1}{|I|} \sum_{k \in I} a_{kj}|), \tag{1}$$

where $\frac{1}{|I|} \sum_{k \in I} a_{kj}$ denotes the average expression value of a gene in I under condition j. The smooth score $s(I, J)$ is actually a refinement of L_∞-distance $d_\infty(,)$, a popular metric in functional analysis. Recall that, for any two n-dimensional vectors $\mathbf{x} = (x_i)$ and $\mathbf{y} = (y_i)$, $d_\infty(\mathbf{x}, \mathbf{y}) = \max_i |x_i - y_i|$. If a gene expression level is considered as a function with condition as variable, clustering process aims to classifying genes into groups each containing genes with similar expression functions. Thus, the smooth score was proposed for gene expression analysis in [24]. If $A(I, J)$ has the smooth score $s(I, J)$, then, for any rows v and v' in $A(I, J)$, $d_\infty(v, v') \leq 2s(I, J)$.

Given a small number $\epsilon > 0$, $A(I, J)$ is an *ϵ-smooth cluster* if $s(I, J) \leq \epsilon$. We formulate the following clustering problem:

Smooth Clustering Problem [24]

Instance: A gene expression matrix $A = (a_{ij})$ with gene set X and condition set Y, a subset $J \subseteq Y$, and a number $\epsilon > 0$;

Question: Find a largest subset $I \subseteq X$ such that $A(I, J)$ is an ϵ-smooth cluster.

In [23], Zhang and Zhu show that the Smooth Clustering problem is NP-hard; therefore, it is desirable to develop efficient approximating algorithms for it. However, this task is difficult too in general. In fact, the proof of the NP-hardness of Smooth Clustering problem (see [23]) gives an approximation-ratio-preservation reduction from the INDEPENDENT SET problem (see [13]). Therefore, there is an $\epsilon > 0$ such that approximating the Smooth Clustering problem within a factor n^ϵ is NP-hard, where n is the number of rows in the input gene expression matrix. In the rest of this section, we shall focus on matrices with only one column, where the Smooth Clustering problem is equivalent to

Smooth Subset Problem: Given a finite set S, a weight $w(s) \geq 0$ for each $s \in S$, and a positive number ϵ, find a largest ϵ-smooth subset $S' \subseteq S$, i.e. $|w(s) - \frac{1}{|S'|} \sum_{t \in S'} w(t)| \leq \epsilon$ for every $s \in S'$.

Zhang and Zhu [23] show that for Smooth Subset problem, there exists a simple $\frac{1}{2}$-approximation algorithm; they also show that there does not exist

an approximation algorithm with ratio better than 0.8 unless NP=P. In the following, we improve this inapproximability result by showing there does not exist an approximation algorithm with ratio better than $\frac{1}{2}$ unless NP=P.

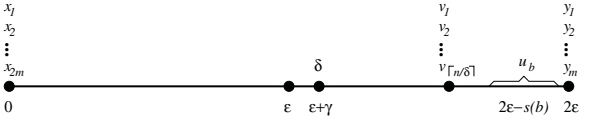
Theorem 4. *Let $k(S, \epsilon)$ be the size of the largest ϵ-smooth subsets of S for any weighted set S and $\epsilon > 0$. For any small constant $\delta > 0$, there is no polynomial-time algorithm that can always output an ϵ-smooth subset of size at least $(\frac{1}{2} + \delta)k(S, \epsilon)$ on an input S unless* NP=P.

Proof. Let δ be a small positive constant. Suppose $\mathcal{A}$ is a polynomial time approximation algorithm with approximation factor $\frac{1}{2} + \delta$ for the Smooth Subset problem. We will show that $\mathcal{A}$ can be used to derive a polynomial time algorithm for the PARTITION problem, contradicting its NP-completeness [10]. Recall that the PARTITION problem is to, given a finite set B and an integer size $s(b) > 0$ for each $b \in B$, decide if there is a subset $B' \subseteq B$ such that $\sum_{b \in B'} s(b) = \sum_{b \in B - B'} s(b)$.

For a weighted set B as an instance of the PARTITION problem, we set

$$\sigma = \sum_{b \in B} s(b), \quad \gamma = (\lceil n/\delta \rceil + 1/2)\sigma, \quad m = n + \lceil n/\delta \rceil.$$

We construct an instance (D, ϵ) of the Smooth Subset problem from B as follows. First, we set $\epsilon = (2m + 3)\sigma$. The set D contains $2m$ x_i's of weight 0, m y_i's of weight 2ϵ and $\lceil n/\delta \rceil$ v_i's of weight $2\epsilon - \sigma$; for each $b \in B$, D contains a unique element u_b of weight $2\epsilon - s(b) > 0$; in addition, D also contains an element z of weight $\epsilon + \gamma$. In total, D contains $4m + 1$ elements as illustrated below.



Fact *If there is a solution to the PARTITION instance B, then D has an ϵ-smooth subset of size at least $2m + 2\lceil n/\delta \rceil + 1$. Otherwise, any ϵ-smooth subset of D has size at most $2m + 1$.*

Proof. Suppose B has a subset B' such that $\sum_{b \in B'} s(b) = \sum_{b \in B - B'} s(b) = \frac{1}{2}\sigma$. Then $D' = \{x_i, y_j \mid i \leq m + \lceil n/\delta \rceil + |B'|,\ j \leq m\} \cup \{v_k, u_b \mid k \leq \lceil n/\delta \rceil,\ b \in B'\} \cup \{z\}$ is a desired ϵ-smooth subset since it contains $2(m + \lceil n/\delta \rceil + |B'|) + 1$ elements and

$$\begin{aligned} &\textstyle\sum_{d \in D'} w(d) \\ &= \textstyle\sum_{j \leq m} w(y_j) + \sum_{k \leq \lceil n/\delta \rceil} w(v_k) + \sum_{b \in B'} w(u_b) + w(z) \\ &= 2m\epsilon + (2\epsilon - \sigma)\lceil n/\delta \rceil + (2\epsilon|B' - \tfrac{1}{2}\sigma) + (\epsilon + \gamma) \\ &= (2(m + \lceil n/\delta \rceil + |B'|) + 1)\epsilon + (\gamma - \sigma(\lceil n/\delta \rceil + \tfrac{1}{2})) \\ &= (2(m + \lceil n/\delta \rceil + |B'|) + 1)\epsilon, \end{aligned}$$

where $w(d)$ denotes the weight of the element d. This proves the first part of the fact.

If there is no solution to the PARTITION instance B, then, $\sum_{b\in B'} s(b) \neq \frac{1}{2}\sigma$ for any subset $B' \subset B$. Let D'' be a largest ϵ-smooth subset of D. Recall that, for each $d \in D$, we use $w(d)$ to denote its weight. Let $\mu = \frac{1}{|D''|}\sum_{d\in D''} w(d)$. We consider the following three cases.

Case 1: $\mu > \epsilon$. Then, D'' does not contain any of the $2m$ elements of weight 0 and hence $|D''| \leq 2m+1$.

Case 2: $\mu < \epsilon$. First, it does not contain any of the y_j's of weight 2ϵ. We further show that either it does not contain more than m elements with weight 0 or it does not contain any of the elements u_b, $b \in B$, and v_k, $1 \leq k \leq \lceil n/\delta \rceil$. This implies that $|D''| \leq 2m+1$.

Assume $|D''| \geq 2m+2$ and there are $m+l$ ($l \geq 1$) elements of weight 0 in D''. Since D'' does not contains any of the elements with weight 2ϵ and $|D''| \geq 2m+2$,

$$\begin{aligned}
\mu &= \tfrac{1}{|D''|}\textstyle\sum_{d\in D''} w(d) \\
&\leq \tfrac{1}{|D''|}[w(z) + \textstyle\sum_{1\leq k\leq \lceil n/\delta\rceil} w(v_k) + \sum_{b\in B} w(u_b)] \\
&= \tfrac{1}{|D''|}[(\epsilon+\gamma) + (2\epsilon-\sigma)\lceil n/\delta\rceil + (2n\epsilon-\sigma)] \\
&= \tfrac{1}{|D''|}[(2m+1)\epsilon - \tfrac{1}{2}\sigma] \\
&\leq \tfrac{2m+1}{2m+2}\epsilon.
\end{aligned}$$

For each $b \in B$, u_b has weight $2\epsilon - s(b)$. Since

$$(2\epsilon - s(b)) - \mu \geq \epsilon - s(b) + \frac{1}{2m+2}\epsilon = \epsilon + (\frac{2m+3}{2m+2}\sigma - s(b)) > \epsilon,$$

u_b is not in D''. Similarly, we can show that all the v_k ($1 \leq k \leq \lceil n/\delta \rceil$) are not in D''.

Case 3: $\mu = \epsilon$. In this case, we have $|D''| \leq 2m$ by proving the fact that D'' does not contain any elements from $\{z, u_b, v_k | b \in B,\ 1 \leq k \leq \lceil n/\delta \rceil\}$. If the fact is not true, we assume D'' contains l more weight-0 elements x_i than weight-(2ϵ) elements y_j. Let $D_1 = D'' \cap \{u_b | b \in B\}$ and $D_2 = D'' \cap \{v_i | 1 \leq k \leq \lceil n/\delta \rceil\}$. We consider the following two subcases.

Case 3.1: $z \notin D''$. Then, since $\mu = \epsilon$, we have

$$(l+|D_1|+|D_2|)\epsilon = \sum_{u_b\in D_1} w(u_b) + \sum_{v\in D_2} w(v) = 2\epsilon(|D_1|+|D_2|) - \sum_{b:u_b\in D_1} s(b) - |D_2|\sigma.$$

This implies $\sum_{b:u_b\in D_1} s(b) + |D_2|\sigma = (|D_1| + |D_2| - l)\epsilon$, a contradiction since the left side is non-zero but smaller than ϵ.

Case 3.2: $z \in D''$. Similarly, we have

$$\begin{aligned}
(1 + l + |D_1| + |D_2|)\epsilon &= \epsilon + \gamma + \textstyle\sum_{u_b\in D_1} w(u_b) + \sum_{v\in D_2} w(v) \\
&= \epsilon + (\lceil \tfrac{n}{\delta} \rceil + \tfrac{1}{2})\sigma + 2\epsilon(|D_1| + |D_2|) \\
&\quad - \textstyle\sum_{b:u_b\in D_1} s(b) - |D_2|\sigma.
\end{aligned}$$

or equivalently $(|D_1| + |D_2| - l)\epsilon = (|D_2| - \lceil n/\delta \rceil)\sigma + (\sum_{b:u_b\in D_1} s(b) - \frac{1}{2}\sigma)$. This implies that $\lceil n/\delta \rceil = |D_2|$ and $\frac{1}{2}\sigma = \sum_{b:u_b\in D_1} s(b)$, contradicting to that there is no solution to the PARTITION instance B. This finishes the proof of the fact.

By assumption, $\mathcal{A}$ is a polynomial time algorithm with approximation factor $\frac{1}{2}+\delta$ for the Smooth Subset problem. Now, we apply $\mathcal{A}$ to the instance D. If $\mathcal{A}$ outputs an ϵ-smooth subset of size at most $2m+1$, then, the largest ϵ-smooth subset has size at most

$$\frac{2m+1}{1/2+\delta} = \frac{2n+2\lceil n/\delta\rceil+1}{1/2+\delta} < 2n + 4\lceil n/\delta\rceil + 1 = 2m + 2\lceil n/\delta\rceil + 1$$

since $\mathcal{A}$ is a $(\frac{1}{2}+\delta)$-approximation algorithm. Thus, we conclude that there is no solution to the PARTITION instance B by the fact proved above. If $\mathcal{A}$ outputs an ϵ-smooth subset of size at least $2m+2$, then, by the fact, there is a solution to the PARTITION instance B. Therefore, we derive a polynomial time algorithm for the PARTITION problem using $\mathcal{A}$, contradicting NP-completeness of the PARTITION problem. □

4 Inferring Plaid Model Problem

With present microarray technology, a gene expression matrix can contain as many as tens of thousands of entries. Therefore, even visualization of a microarray data is challenging. One natural way to do this is to first form a color image of the data $A = (a_{ij})$ on an $|X|$ by $|Y|$ grid, with each cell colored according to the value of a_{ij}. Then, it proceeds with reordering the rows and columns so that similar rows and columns are grouped together and hence an image with blocks of similar color is formed. For instance, the rows and columns can be reordered after running a hierarchical clustering method on genes [9].

An ideal reordering of the array would produce an image with K rectangular blocks each being nearly uniformly colored. Mathematically, this ideal corresponds to the existence of a disjoint K'-partition of genes and a disjoint K''-partition of conditions such that $K'K'' = K$ and

$$a_{ij} = c_0 + \sum_{k'=1}^{K'} \sum_{k''=1}^{K''} \lambda_{k'k''}(i,j) c_{k'k''}$$

where c_0 is a background color, $c_{k'k''}$ is the color in the block specified by the k'th gene-block $X_{k'}$ and the k''th condition block $Y_{k''}$, and $\lambda_{k'k''}(i,j)$ is 1 if $i \in X_{k'}$ and $j \in Y_{k''}$, and 0 otherwise.

However, it is more likely that the blocks will overlap in some places in real data. By removing the constraints that K color blocks are disjoint, we obtain the *plaid model* that represents the microarray data as a sum of possible overlapping 'constant' layers. Algebraically, the plaid model corresponds to the decomposition of A into K 'uniform' matrices $B_k = (b_{ij})$'s that are defined over gene subset X_k and condition subset Y_k such that $a_{ij} = \sum_{k=1}^{K} c'_k(i,j)$, where $c'_k(i,j) = b_{ij}$ if $i \in X_k$ and $j \in Y_k$, and 0 otherwise. Let ρ_{ik} be 1 if i is in the gene subset X_k and 0 otherwise. To capture biological interests of identifying genes that had identically, though not constantly, co-expressed in a subset of conditions, the model allows each matrix $B_k = (b_{ij})$ takes one of the following forms ([17]):

$$b_{ij} = c_k$$
$$b_{ij} = c_k + \alpha_{ik}$$
$$b_{ij} = c_k + \beta_{kj}$$
$$b_{ij} = c_k + \alpha_{ik} + \beta_{jk}$$

where if α_{ik} is used, we request that $\sum_i \rho_{ik}\alpha_{ik} = 0$ to avoid overparameterization, with similar condition on β_{jk}.

The clustering problem under the plaid model is to seek a model that best fits the data, i.e. with a smallest value of

$$\sum_i \sum_j [a_{ij} - \sum_{k=1}^{K} c'_k(i,j)]^2.$$

In this paper, we study the following decision version of the above clustering problem. For simplicity, we say a matrix to be *uniform* if all its entries are identical. Furthermore, we use (b) to denote the uniform matrix whose entries are b.

Plaid Model Fitting
Instance: A gene expression matrix $A = (a_{ij})$ with gene set X and condition set Y, and positive integer $K \leq |X||Y|$.
Question: Are there K uniform submatrices $B_k = (b_k)$ $(b_k \geq 0,\ 1 \leq k \leq K)$ with gene subset X'_i and condition subset Y'_i such that for each i and j, $a_{ij} = \sum_{k=1}^{K} \lambda_k(i,j)b_k$, where $\lambda_k(i,j)$ is 1 if $i \in X'$ and $j \in Y'$, and 0 otherwise?

Theorem 5. *The Plaid Model Fitting problem is NP-complete.*

Proof. . Here we only give the sketch of the proof due to the page limitation. A complete proof can be found in the full version of this work. We prove the above theorem via a reduction from the following graph decomposition problem:
Complete Bipartite Subgraph Decomposition
Instance: A weighted bipartite graph $G = (V, E)$ in which each edge has a positive weight, and integer K.
Question: Can G be decomposed into K positively-weighted completed bipartite graphs G_i such that all the edges in G_i have same weight, and for each edge in E, its weight in G is equal to the sum of its weights in G_i's.

Such a reduction is more or less straightforward since a matrix $A = (a_{ij})$ with non-negative matrix gives a unique weighted bipartite graph, in which the edge from the vertex corresponding to row i to the vertex corresponding to column j is assigned weight a_{ij}. □

References

1. A. Alizadeh *et al.*, Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling, *Nature* 403(2000), 503-510.
2. G. Ausiello *et al.*, *Complexity and Approximation*, Springer Verlag, 1999.
3. A. Ben-Dor, Z. Yakhini. Clustering gene expression patterns. In *Proc. RECOMB'99*, 33-42.

4. A. Ben-Dor, B. Chor, R. Karp, and Z. Yakhini, Discovering local structure in gene expression data: The order-preserving submatrix problem. In *Proceedings of RECOMB'02*, 49-57.
5. P. Berman, B. DasGupta, S. Muthukrishnan and S. Ramaswami, Efficient approximation algorithm for tiling and packing problems with rectangles. *J. Alg.* 41(2001), 443-470.
6. Y. Chen, E. Dougherty, M. Bitter, Ratio-based decisions and the quantitative analysis of cDNA microarray images. *J. Biomed. Optics* 2(1997), 364-374.
7. Y. Cheng and G. Church, Biclustering of expression data, In *Proceedings of ISMB'2000*, 93-103.
8. T. H. Cormen *et al. Introduction to Algorithms* (2nd Ed.), McGraw-Hill, 2001.
9. M.B. Eisen *et al.*, Clustering Analysis and display of genome-wide expression pattern. *Proc. Natl. Amer. Sci.* 95(1998), 14863-68.
10. M. R. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. San Francisco: Freeman, 1979.
11. E. Hartuv *et al.*, An algorithm for clustering cDNAs for gene expression analysis, In *Proceedings of Recomb'99*, 188-197.
12. I. Hedenfalk *et al. Gene-expression profiles in hereditary breast cancer*, New England Journal of Medicine, 344:539-548, 2001.
13. D.S. Hochbaum, *Approximation Algorithms for NP-hard Problems*, PWS Publishing Co., 1995.
14. T.G. Kolda and D.P. O'Leary, A semidiscrete matrix decomposition for latent semantic indexing in information retrieval, *ACM Trans. on Information Systems* 16(1998), 322-346.
15. E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston Inc., 1976.
16. J. Liu, J. Yang and W. Wang, *Biclustering in gene expression data by tendency*, in *Proceedings of CSB'04*, 182-193, 2004.
17. L. Lazzeroni and A. Owen, Plaid Models for Gene Expression Data, *Statistica Sinica* 12(2002), 61-86. See *http://www-stat.stanford.edu/∼owen* for more about Plaid model.
18. D.D. Lee and H.S. Seung, Learning the parts of objects by non-negative matrix factorization, *Nature* 401(1999), 788-791.
19. R. Peeters, The maximum edge biclique problem is NP-complete, *Discrete Applied Mathematics* 131 (2003), 651-654.
20. P. Tamayo *et al.*, Interpreting patterns of gene expression with self-organizing maps: methods and application to hematopoietic differentiation. *Proc. Natl. Acad. Sci.* 96(1999), 2907-12.
21. O. Troyanskaya *et al.*, Missing value estimation methods for DNA microarrays. *Bioinformatics* 17(2001), 520-525.
22. M. Yannakakis, Node-and edge-deletion NP-complete problems, in *Proceedings of the 10th Annual STOC*, 253-264, 1978.
23. L. Zhang and S. Zhu, Complexity Study on Two Clustering Problems. In *Proceedings of the Annual Inter. Symposium on Alg. and Comput.*, 660-669, 2001.
24. L. Zhang and S. Zhu, A new approach to clustering gene expression data. In *Proceedings of IEEE Symposium on Bioinformatics*, 268-275, 2002.

RIATA-HGT: A Fast and Accurate Heuristic for Reconstructing Horizontal Gene Transfer

Luay Nakhleh[1], Derek Ruths[1], and Li-San Wang[2]

[1] Department of Computer Science, Rice University
Houston, TX 77005, USA
{nakhleh,druths}@cs.rice.edu
[2] Department of Biology, University of Pennsylvania
Philadelphia, PA 19104, USA
lswang@mail.med.upenn.edu

Abstract. Horizontal gene transfer (HGT) plays a major role in microbial genome diversification, and is claimed to be rampant among various groups of genes in bacteria. Further, HGT is a major confounding factor for any attempt to reconstruct bacterial phylogenies. As a result, detecting and reconstructing HGT events in groups of organisms has become a major endeavor in biology. The problem of detecting HGT events based on incongruence between a species tree and a gene tree is computationally very hard (NP-hard). Efficient algorithms exist for solving restricted cases of the problem.

We propose RIATA-HGT, the first polynomial-time heuristic to handle all HGT scenarios, without any restrictions. The method accurately infers HGT events based on analyzing incongruence among species and gene trees. Empirical performance of the method on synthetic and biological data is outstanding. Being a heuristic, RIATA-HGT may overestimate the optimal number of HGT events; empirical performance, however, shows that such overestimation is very mild.

We have implemented our method and run it on biological and synthetic data. The results we obtained demonstrate very high accuracy of the method. Current version of RIATA-HGT uses the PAUP tool, and we are in the process of implementing a stand-alone version, with a graphical user interface, which will be made public. The tool, in its current implementation, is available from the authors upon request.

1 Introduction

Horizontal (also known as lateral) gene transfer (HGT) plays a major role in microbial genome diversification [7, 20], and is claimed to be rampant among various groups of genes in bacteria [8]. M.-W. Ho also has written of the risks that HGT poses to humans, which include (1) antibiotic resistance genes spreading to pathogenic bacteria; (2) disease-associated genes spreading and recombining to create new viruses and bacteria that cause diseases; and (3) transgenic DNA inserting into human cell, triggering cancer [11]. Furthermore, the occurrence of HGT confounds or completely defeats any attempt to reconstruct evolution (especially for bacterial organisms) as has been famously summarized by Ford Doolittle [8]

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 84–93, 2005.

> Molecular phylogeneticists will have failed to find the "true tree," not because their methods are inadequate or because they have chosen the wrong genes, but because the history of life cannot properly be represented as a tree.

In light of all this evidence supporting the significance of HGT as an evolutionary mechanism, its risks, and its confounding effects on evolution reconstruction, much research in biology today is dedicated to the problems of understanding the nature of HGT events and detecting and reconstructing them (their numbers as well as their donors and recipients).

In order to reconstruct genomic changes, what we usually seek is the species (organismal) phylogeny – the tree that traces the history of the replicating cell lineages that transmit genes and genomes to successive generations [15]. This species phylogeny provides the backdrop against which events such as HGT have occurred. In their study, Lerat *et al.* identified a set of genes resistant to HGT (those, according to the authors, are usually single-copy orthologous genes), combined them and built a species tree [15]. They tested whether a given gene had been horizontally transferred by comparing its tree (topology) against the species trees. Based on this study, the problem of detecting and reconstructing HGT is formulated as follows: given a species tree ST and a set G of gene trees, compute the minimum-cardinality set of HGT events whose occurrence on tree ST give rise to the gene trees in G (we give a mathematical definition of the problem in Section 2.1). The problem's formulation as an optimization problem, in which a minimum-cardinality set of HGT events is sought, is a reflection of Occam's razor: in the absence of any additional biological knowledge, HGT events should be used sparingly to explain data features otherwise explainable under a tree model. Further, the actual set of HGT events may not be computationally identifiable in certain cases since multiple (equally optimal) solutions may exist for the problem. The problem of finding a minimum-cardinality set of HGT events whose occurrence of species tree ST would give rise to the gene trees in set G is computationally NP-hard [4]. Efficient solutions for the problem exist, but for limited special cases [10, 19].

In this paper, we propose a polynomial-time method, RIATA-HGT, for solving a relaxed version of the problem, where we drop the optimality criterion in the problem definition. Although the cardinality of the HGT set computed by our method is not guaranteed (theoretically) to be the minimum (among all such sets), experimental results of our method, on both biological and synthetic data, demonstrate that, in practice, the method almost always infers the correct set of HGT events. Whenever the method overestimates the minimum amount of HGT events, such an overestimation is very mild (often one or two additional HGT events). RIATA-HGT takes as input a species tree and a set of gene trees, and computes HGT events to explain all of those gene trees.

The rest of the paper is organized as follows. In Section 2 we briefly review the biology behind HGT, and give an overview of the techniques and tools for analyzing it. In Section 2.1 we mathematically define the problem of species-tree/gene-tree incongruence and HGT reconstruction. Section 3 describes RIATA-HGT and the underlying algorithms Further, we analyze the theoretical properties of the method. In Section 4 we show the performance of our method on biological as well as synthetic data. We conclude in Section 5 with final remarks and directions for future research.

2 Horizontal Gene Transfer and Its Detection: A Brief Overview

In horizontal gene transfer (HGT), genetic material is transferred from one lineage to another, such that certain sites (specified by a specific substring within the DNA sequence of the species into which the horizontally transferred DNA was inserted) are inherited through horizontal transfer from another species, while all others are inherited from the parent. Horizontal transfers are believed to be ubiquitous among bacteria and still quite common in other branches of the Tree of Life [7] – although this view has recently been challenged [9, 13, 23, 24]. The three major modes of HGT in the Archaea and Bacteria are *transformation* (uptake of naked DNA from the environment), *conjugation* (transfer of DNA by direct physical interaction between a donor and a recipient), and *transduction* (transfer of DNA by phage infection) [21].

The goal of much biological research has been to identify those genes that were acquired by the organism through horizontal transfers rather than inherited from its ancestors. In one of the first papers on the topic, Medigue [17] proposed the use of multivariate analysis of codon usage to identify such genes; since then various authors have proposed other intrinsic methods, such as using GC content, particularly in the third position of codons (e.g., [14]). An advantage of intrinsic approaches is their ability to identify and eliminate genes that do not obey a tree-like process of evolution and thus could prevent classical phylogenetic methods from reconstructing a good tree. With the advent of whole-genome sequencing, more powerful intrinsic methods become possible, such as the location of suspect genes with each genome: such locations tend to be preserved through lineages, but a transfer event can place the new gene in a more or less random location. Thus, biologists often consider the neighbors of a gene in the prokaryotic genome to identify horizontal transfers. However, differential selection pressure, uneven evolutionary rates, and biased gene sampling can all give rise to false identification of HGT [9].

Non-intrinsic approaches involve phylogenetic reconstruction. These methods use phylogenetic reconstructions to identify discrepancies that could tag transfer events. An old question in phylogenetic reconstruction has been "to combine or not to combine?" – that is, given DNA sequences for several genes, are we better off concatenating the sequences or analyzing each set separately (e.g., [6])?

The common sense conclusion that many genes inherited through lineal descent would override the confusing signal generated by a few genes acquired through horizontal transfer appears wrong [5, 27]. Of course, one must first resolve the old problem of gene trees vs. species trees: discrepancies between the trees derived from different genes do not necessarily indicate reticulate evolution, but may simply testify to the incongruent evolution of two or more genes, all within a valid, tree-shaped evolution of the species (e.g., [16, 22]). Distinguishing between the two is difficult in the absence of additional information.

With whole-genome sequencing, such information becomes available. Huynen [12] advocate two types of data: the fraction of shared orthologs and gene synteny. Synteny (the conservation of genes on the same chromosome) is not widely applicable with prokaryotes, but its logical extension, conservation of gene order, definitely is – and Huynen and Bork proposed to measure the fraction of conserved adjacencies.

2.1 The Graph-Theoretic Approach

From a graph-theoretic point of view, the problem can be formulated as pure phylogenetic network reconstruction [18, 19]. In the case of HGT, a phylogenetic network is a pair (T, Ξ), where T is the species (organismal) tree, and Ξ is a set of HGT edges whose addition to T creates a directed acyclic graph (DAG) N, referred to as a phylogenetic network. We say that a network N (interchangeably, pair (T, Ξ)) induces a tree T' if T' can be obtained from N by the following two steps: (1) for each node in N that has two edges coming into it, remove one of the two edges, and (2) suppress each node that has one incoming edge and one outgoing edge.

The problem of resolving incongruence between a species tree ST and a set G of gene trees is then defined as follows.

Problem 1. (The HGT Reconstruction Problem)

> **Input:** A species tree ST and a set G of gene trees.
> **Output:** Set Ξ of minimum cardinality such that the pair (ST, Ξ) induces each of the gene trees in G.

As mentioned before, the minimization criterion reflects the fact that the simplest solution is sought; in this case, the simplest solution is one with the minimum number of HGT events. Hallett and Lagergren [10] gave an efficient algorithm for solving the HGT Reconstruction Problem; however, their algorithm handles limited special cases of the problem in which the number of HGT events is very small, and the number of times a gene is transferred is very low (also, their tool handles only binary trees; [2]). Boc and Makarenkov [3] solve the problem by reconstructing the species tree from a sequence alignment, and then add edges to the tree to minimize a distance-based optimization criterion. Since the original alignment contains HGT events, the starting tree may be inaccurate, which results in the addition of arbitrary HGT events. Further, distance-based approaches suffer from a lack of accurate techniques for estimating branch lengths. Nakhleh *et al.* [19] gave efficient algorithms for solving the problem, but for limited special cases referred to as gt-networks; further, they handled only binary trees. In the next section, we describe our method, RIATA-HGT, which is a heuristic for solving the HGT Reconstruction Problem, and demonstrate its empirical performance in Section 4. RIATA-HGT is the first method for solving the general case of the HGT Reconstruction Problem.

3 RIATA-HGT

3.1 Terminology and Notation

A rooted phylogenetic tree T over set S of taxa is a rooted tree with $|S|$ leaves, each labeled by a unique element of S. We denote by $r(T)$ the root of T and by $L(T)$ the leaf set of T. Let T be a rooted phylogenetic tree over set S of taxa, and let $S' \subseteq S$. We denote by $T(S')$ the minimal rooted subtree of T that connects all the element of S'. Furthermore, the restriction of T to S', denote $T|S'$ is the rooted subtree that is obtained from $T(S')$ by suppressing all vertices (except for the root) whose number of incident

edges is 2. Let S' be a maximum-cardinality set of leaves such that $T_1|S' = T_2|S'$, for two trees T_1 and T_2; we call $T_1|S'$ (equivalently, $T_2|S'$) the maximum agreement subtree of the two trees, denoted $MAST(T_1, T_2)$. A *clade* of a tree T is a complete subtree of T. Let $T' = MAST(T_1, T_2)$; then, $T_1 - T'$ is the set of all maximal clades whose pruning from T_1 yields T' (we define $T_2 - T'$ similarly). In other words, there do not exist two clades u and u' in $T_1 - T'$ such that either u is a clade in u', or u' is a clade in u.

We say that node x reaches node y in tree T if there is a directed path from x to y in T. We denote the root of a clade t by $r(t)$. We say that clade t_1 reaches clade t_2 (both in tree T) if $r(t_1)$ reaches $r(t_2)$. The sibling of node x in tree T is node y, denoted $sibling_T(x) = y$ whenever x and y are children of the same node in T. We denote by T_x the clade rooted at node x in T. The least common ancestor of a set X of taxa in tree T, denoted $lca_T(X)$ is the root of the minimal subtree of T that contains the leaves of X. The edge incoming into node x in tree T is denoted by $inedge_T(x)$.

3.2 The Algorithm

We describe the algorithm underlying RIATA-HGT in terms of a species tree and a gene tree. Our implementation of RIATA-HGT allows the user to specify a set of gene trees, and it iterates over each pair of the species tree and a gene tree, and summarizes the results for all trees. The core of RIATA-HGT is the divide-and-conquer algorithm ComputeHGT algorithm (outlined in Fig. 1). The algorithm starts by computing the $MAST$, T', of the species tree ST and gene tree GT; tree T' forms the basis for detecting and reconstructing the HGT events (computing T' is done in Step 1 in Fig. 1). The algorithm then decomposes the clade sets U_1 and U_2 (whose removal from ST and GT, respectively, yields T') into maximal clades such that each maximal clade in one of the two sets is "matched" by a maximal clade on the same leaf set in the second set. The algorithm for this decomposition is outlined in Fig. 2. The algorithm then recurses on each maximal clade and its matching maximal clade (Steps 5.c.(1) and 5.d.(5).(1) in Fig. 1) to compute the HGT events whose recipients form sub-clades of those maximal clades. Finally, we add a single HGT event per each maximal clade to connect it to its "donor" in the ST; this is achieved through the calls to AddSingleHGT (Fig. 3) in Steps 5.c.(2) and 5.d.(5).(3) in Fig. 1. We have the following properties of the method, which we present without proofs due to space constraints: (1) ComputeHGT always terminates, (2) The pair (ST, Ξ) (the output of ComputeHGT) induces the tree GT, and (3) $ComputeHGT$ takes $O((h^2 + \log n)n^2)$ time.

4 Experimental Results and Discussion

We have implemented RIATA-HGT using the Python language [1], and used the PAUP* tool [26] for computing the maximum agreement subtree of two trees. We studied the empirical performance of our method on synthetic as well as biological datasets. We tried to run the tool of [10], but, unfortunately, the program crashed on almost all datasets.

To obtain synthetic datasets, we have written a simulator that takes a (species) tree ST as input, and adds a randomly generated set Ξ of HGT events to T, where the

PROCEDURE COMPUTEHGT(ST,GT)
Input: Species tree ST, and gene tree GT, both on the same set S of taxa.
Output: Computes the set Ξ of HGT events such that the pair (ST, Ξ) induces GT.

1. $T' = MAST(ST, GT)$;
2. **If** $T' = ST$ **then**
   (a) **Return**;
3. $U_1 = ST - T'$; $U_2 = GT - T'$;
4. $V = \emptyset$;
5. **Foreach** $u_2 \in U_2$
   (a) $Decompose(U_1, u_2, T', V)$;
6. $U_2 = V$;
7. **While** $V \neq \emptyset$
   (a) Let u_2 be an element of V;
   (b) Let $u_1 \in U_1$ be such that $L(u_2) \subseteq L(u_1)$;
   (c) $Y = \{y \in U_2 : L(y) \cap L(u_1) \neq \emptyset\}$;
   (d) $Z = \{y | (L(y) - L(u_1)) : \ y \in Y\}$;
   (e) $V = V - Y$; $V = V \cup Z$;
   (f) $X = \{u_1 | L(y) : y \in Y\}$;
   (g) **Foreach** $y \in Y$
      i. Let $x \in X$ be such that $L(x) \cap L(y) \neq \emptyset$;
      ii. $ComputeHGT(x, y)$;
      iii. $AddSingleHGT(ST, GT, y, U_2, T')$;

Fig. 1. The main algorithm for detecting and reconstructing HGT events based on a pair of species tree and gene tree.

number of events in Ξ is specified by the user. The simulator also implements certain constraints so that the network N resulting from adding Ξ to ST is a directed acyclic graph. Once the pair (ST, Ξ) is generated, the simulator outputs a (gene) tree GT that uses all the edges in Ξ. In other words, GT results from N by

1. for each node v in N such that there are two edges incoming into v, remove the edge that is not in Ξ; and
2. suppress all nodes that has only one incoming edge and one outgoing edge.

To generate a species tree, ST, we used the r8s tool [25], which generates random birth-death trees, with a number of leaves specified by the user.

We studied the performance of RIATA-HGT on ten different sizes k of Ξ, where k ranges from 1 to 10. For each k, we generated 30 triplets (ST, Ξ, GT), with $|\Xi| = k$, as described above, and ran RIATA-HGT on (ST, GT). We studied the performance of RIATA-HGT in terms of the number of HGT events it predicts compared to the actual number of HGT events. We looked at the predictions in each of the runs (Fig. 4(a)), and at the averages (Fig. 4(b)).

The box-and-whisker plot in Fig. 4(a) show that RIATA-HGT computed the correct set of HGT events in most cases. In particular, RIATA-HGT obtained the exact number of HGT events (with the exception of very few outliers) in the cases when

PROCEDURE DECOMPOSE(U_1,u_2, T, U')
Input: Set U_1 of clades from ST, clade u_2 from GT, the backbone clade u_2, and U' which will contain the "refined" clades of u_2.
Output: Decompose u_2 so that no clade in U' has a leaf set that is the union of leaf sets of more than one clade in U_1.

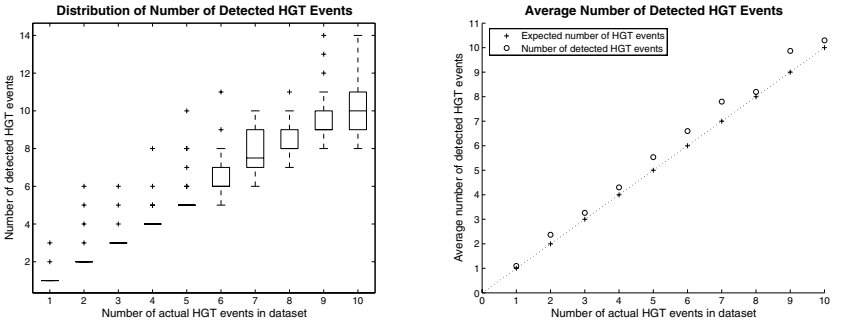
1. **If** $\exists u_1 \in U_1$ such that $L(u_2) \subseteq L(u_1)$ **then**
   (a) $U' = U' \cup \{u_2\}$;
   (b) $B(u_2) = T$;
   (c) **Return** u_2;
2. **Else**
   (a) **If** $\exists u_1 \in U_1$ such that $r(u_2) = r(u_2|L(u_1))$
      i. $t = u_2|L(u_1)$;
      ii. $U' = U' \cup \{t\}$;
      iii. $B(t) = T$;
      iv. Let $X = u_2 - t$;
      v. **Foreach** $x \in X$
         A. $Decompose(U_1, x, t, U')$;
      vi. **Return** t;
   (b) **Else**
      i. Let $c_1, \ldots, c_k$ be the children of $r(u_2)$;
      ii. $x = Decompose(U_1, T_{c_1}, T, U')$;
      iii. **For** $i = 2$ to k
         A. $Decompose(U_1, T_{c_i}, x, U')$;
      iv. **Return** x;

Fig. 2. The algorithm for decomposing the clades in U_1 and U_2 such that for all $u_1 \in U_1$ and $u_2 \in U_2$ we have $L(u_1) \not\subset L(u_2)$.

PROCEDURE ADDSINGLEHGT(ST, GT, u_2, U_2, T')
Input: Species tree ST, gene tree GT, clade u_2 of GT, set U_2 of clades of GT, and $MAST$ T' of ST and GT.
Output: Add to Ξ a single HGT event whose donor is determined in this procedure and whose recipient is clade u_2.

1. $Q = L(u_2) \cup L(B(u_2))$;
2. $T'' = GT|Q$; $\quad p = lca_{T''}(L(u_2))$;
3. $tq = lca_{ST}(L(u_2))$; $\quad te = inedge_{ST}(tq)$;
4. **If** p is a child of $r(T'')$ and $|L(B(u_2))| > 1$ **then**
   (a) $sq = lca_{ST}(L(B(u_2)))$;
   (b) $\Xi = \Xi \cup (sq \rightarrow te)$;
5. **Else**
   (a) $O = \bigcup_{\{p':p'=sibling_{T''}(p)\}} L(T_{p'})$;
   (b) $sq = lca_{ST}(O)$; $\quad se = inedge_{ST}(sq)$;
   (c) $\Xi = \Xi \cup (se \rightarrow te)$;

Fig. 3. The algorithm for detecting and reconstructing the single HGT event in which clade u_2 is the recipient.



(a) (b)

Fig. 4. (a) A box-and-whisker plot for the predictions of HGT event numbers made by RIATA-HGT. (b) The averages of HGT event numbers estimated by RIATA-HGT vs. the actual number of HGT events.

the actual number of HGT events was between 1 and 5. For datasets with 6 to 10 HGT events, RIATA-HGT predicted the correct number in a large number of the cases. When RIATA-HGT underestimated the actual number, the method found the minimum number of HGT events, as opposed to the actual number (for reasons outlined in Section 2.1). In the absence of any additional biological knowledge, computing the minimum number of such events amounts to finding the simplest solution. The cases where RIATA-HGT overestimates the number of HGT events are a reflection of the heuristic nature of the method. Nonetheless, the overestimation is very mild, as Fig. 4(a) shows. In Fig. 4(b), we plot the average predictions of RIATA-HGT versus the expected number of HGT events in the input. The figure shows a very mild deviation of the average predicted numbers of HGT events from the expected numbers.

For the biological dataset, we considered the species tree and two gene trees of the γ-Proteobacteria group, as reported in [15]. The species tree (shown in Figs. 5(a) and 5(b)) was reconstructed by the authors using a phylogenetic analysis on a sequence dataset obtained by concatenating 203 orthologous gene datasets, all of whose gene trees were concordant. Fig. 5(c) shows the gene tree of the biotin synthase enzyme (BioB), and Fig. 5(d) shows the gene tree of the virulence factor MviN. Both gene trees are incongruent with the species tree of Fig. 5(a).

RIATA-HGT computed the set Ξ of HGT events shown in Fig. 5(a) when invoked on the species tree and the biotin gene tree, and the set Ξ of HGT events shown in Fig. 5(b) when invoked on the species tree and the virulence gene tree. Those two sets of HGT events were hypothesized in [15], and RIATA-HGT computed them. In summary, RIATA-HGT performed very well on the synthetic datasets we generated, as well as on the biological dataset we used.

5 Conclusions and Future Work

We proposed a new method, RIATA-HGT, for detecting and reconstructing horizontal gene transfer events. Our method is a polynomial-time heuristic that, given a species

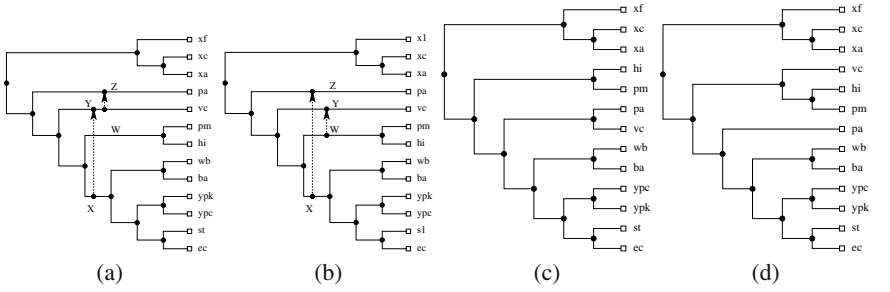


Fig. 5. The species tree of the γ-Proteobacteria group, as reported in [15], is shown by the solid lines in (a) and (b). The two HGT events were computed by RIATA-HGT using the gene tree in (c) are shown by the dotted arrows in (a), and the two HGT events were computed by RIATA-HGT using the gene tree in (d) are shown by the dotted arrows in (b).

tree and a set of gene trees as input, attempts to compute the set of HGT events that explain all the gene trees. Despite the lack of theoretical bounds on the performance of our method, we demonstrated, using synthetic and biological datasets, that RIATA-HGT has excellent performance in practice. RIATA-HGT is the first fast heuristic, with proven empirical performance, that handles the general case of the HGT Reconstruction Problem.

We plan to provide a standalone version of RIATA-HGT (along with a graphical user interface) to the research community. Future work includes testing the method on more biological datasets, more efficient handling of multiple gene trees (instead of the iterative process currently implemented), and extending the method to handle cases in which not all homologs of genes are found across all species.

References

1. Python software foundation, 2005. www.python.org.
2. L. Addario-Berry, M.T. Hallett, and J. Lagergren. Towards identifying lateral gene transfer events. In *Proc. 8th Pacific Symp. on Biocomputing (PSB03)*, pages 279–290, 2003.
3. A. Boc and V. Makarenkov. New efficient algorithm for detection of horizontal gene transfer events. In *Proc. 3rd Int'l Workshop Algorithms in Bioinformatics (WABI03)*, volume 2812, pages 190–201. Springer-Verlag, 2003.
4. M. Bordewich and C. Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics*, pages 1–15, 2005. In press.
5. J.R. Brown, C.J. Douady, M.J. Italia, W.E Marshall, and M.J. Stanhope. Universal trees based on large combined protein sequence data sets. *Nat. Genet.*, 28:281–285, 2001.
6. J.J. Bull, J.P. Huelsenbeck, C.W. Cunningham, D. Swofford, and P. Waddell. Partitioning and combining data in phylogenetic analysis. *Syst. Biol.*, 42(3):384–397, 1993.
7. F. de la Cruz and J. Davies. Horizontal gene transfer and the origin of species: lessons from bacteria. *Trends Microbiol.*, 8:128–133, 2000.
8. W.F. Doolittle. Phylogenetic classification and the universal tree. *Science*, 284:2124–2129, 1999.
9. J.A. Eisen. Horizontal gene transfer among microbial genomes: New insights from complete genome analysis. *Curr Opin Genet Dev.*, 10(6):606–611, 2000.

10. M.T. Hallett and J. Lagergren. Efficient algorithms for lateral gene transfer problems. In *Proc. 5th Ann. Int'l Conf. Comput. Mol. Biol. (RECOMB01)*, pages 149–156, New York, 2001. ACM Press.
11. M.-W. Ho. Recent evidence confirms risks of horizontal gene transfer, 2002. http://www.i-sis.org.uk/FSAopenmeeting.php.
12. M.A. Huynen and P. Bork. Measuring genome evolution. *Proc. Nat'l Acad. Sci., USA*, 95:5849–5856, 1998.
13. C.G. Kurland, B. Canback, and O.G. Berg. Horizontal gene transfer: A critical view. *Proc. Nat'l Acad. Sci., USA*, 100(17):9658–9662, 2003.
14. J.G. Lawrence and H. Ochman. Amelioration of bacterial genomes: rates of change and exchange. *J. Mol. Evol.*, 44:383–397, 1997.
15. E. Lerat, V. Daubin, and N.A. Moran. From gene trees to organismal phylogeny in prokaryotes: The case of the γ-proteobacteria. *PLoS Biology*, 1(1):1–9, 2003.
16. W. Maddison. Gene trees in species trees. *Syst. Biol.*, 46(3):523–536, 1997.
17. C. Medigue, T. Rouxel, P. Vigier, A. Henaut, and A. Danchin. Evidence for horizontal gene transfer in E. coli speciation. *J. Mol. Biol.*, 222:851–856, 1991.
18. B.M.E. Moret, L. Nakhleh, T. Warnow, C.R. Linder, A. Tholse, A. Padolina, J. Sun, and R. Timme. Phylogenetic networks: modeling, reconstructibility, and accuracy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):13–23, 2004.
19. L. Nakhleh, T. Warnow, and C.R. Linder. Reconstructing reticulate evolution in species–theory and practice. In *Proc. 8th Ann. Int'l Conf. Comput. Mol. Biol. (RECOMB04)*, pages 337–346, 2004.
20. H. Ochman, J.G. Lawrence, and E.A. Groisman. Lateral gene transfer and the nature of bacterial innovation. *Nature*, 405(6784):299–304, 2000.
21. P.J. Planet. Reexamining microbial evolution through the lens of horizontal transfer. In R. DeSalle, G. Giribet, and W. Wheeler, editors, *Molecular Systematics and Evolution: Theory and Practice*, pages 247–270. Birkhauser Verlag, 2002.
22. A. Rokas, B.L. Williams, N. King, and S.B. Carroll. Genome-scale approaches to resolving incongruence in molecular phylogenies. *Nature*, 425:798–804, 2003.
23. S.L. Salzberg and J.A. Eisen. Lateral gene transfer or viral colonization? *Science*, 293:1048, 2001.
24. S.L. Salzberg, O. White, J. Peterson, and J.A. Eisen. Microbial genes in the human genome – lateral transfer or gene loss? *Science*, 292(5523):1903–1906, 2001.
25. M. Sanderson. `r8s` software package. Available from http://loco.ucdavis.edu/r8s/r8s.html.
26. D.L. Swofford. PAUP*: Phylogenetic analysis using parsimony (and other methods), 1996. Sinauer Associates, Underland, Massachusetts, Version 4.0.
27. S.A. Teichmann and G. Mitchison. Is there a phylogenetic signal in prokaryote proteins? *J. Mol. Evol.*, 49:98–107, 1999.

A New Pseudoknots Folding Algorithm for RNA Structure Prediction*

Hengwu Li[1,2] and Daming Zhu[1]

[1] School of Computer Sci and Tech, Shan Dong Univ, Jinan 250100, P.R. China
hengwuli@mail.sdu.edu.cn, dmzhu@sdu.edu.cn
[2] Department of Computer, Shan Dong Economic Univ, Jinan 250014, P.R. China

Abstract. A new dynamic programming algorithm with $O(n^4)$ time and $O(n^3)$ space is presented to predict the RNA secondary structure including nested pseudoknots and a subclass of crossed pseudoknots. Compared with the Jens algorithm of $O(n^4)$ time and $O(n^2)$ space, this algorithm can predict more complex pseudoknots. Compared with the Rivas algorithm of $O(n^6)$ time and $O(n^4)$ space, this algorithm has the same power for the planar pseudoknots prediction.

1 Introduction

Energy minimization method for RNA secondary structure predication appears very useful and interests many researchers in recent years. The Zuker algorithm is used to predict the secondary structure without pseudoknots. This algorithm requires $O(n^3)$ time and $O(n^2)$ space for a sequence of length n and is implemented by MFOLD[1] and ViennaRNA[2] programs.

Pseudoknots are functionally important in several known RNAs[3]. Finding the best structure including arbitrary pseudoknots has been proved to be NP-hard[4]. One method for folding RNA pseudoknots adopts heuristic search, such as quasi-Monte Carlo searches[5] and genetic algorithms[6]. Another method resorts to more simplistic energy model to fold RNA pseudoknots, such as the $O(n^4)$ time and $O(n^3)$ space algorithm for base pair maximization[7], and the $O(n^3)$ time algorithm for maximum weight matching[8].

Recently, the method for folding a tractable subclass of pseudoknots under the established thermodynamic model demonstrates more vital forces. The Rivas algorithm can recognize the class of arbitrary planar and restricted non-planar pseudoknots in $O(n^6)$ time and $O(n^4)$ space[3]. Further improvements have been shown to be possible for more restricted classes. For example, an $O(n^5)$ time and $O(n^3)$ space algorithm only considers the class of one planar pseudoknot[4], the $O(n^5)$ time and $O(n^4)$ space algorithms consider the class of arbitrary planar pseudoknots and partial non-planar pseudoknot[9, 10], and an $O(n^4)$ time and $O(n^2)$ space algorithm only considers the class of simple nested pseudoknots[11].

In this paper, a new dynamic programming algorithm with $O(n^4)$ time and $O(n^3)$ space is presented to predict the RNA secondary structure including pla-

* The work is supported by NSFC: 60273032.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 94–103, 2005.

nar pseudoknots and restricted crossed pseudoknots. Our programming experiment confirms that the algorithm can compute 252 of 253 pseudoknots in the PseudoBase[12]. Two new structures, semi-extensible structures and k-stems, are added to the computational model of the RNA folding in [11]. The new model can represent any nested pseudoknots and part of crossed pseudoknots. Thus we can design a dynamic programming algorithm to implement the computation of the model in $O(n^4)$ time and $O(n^3)$ time.

2 Algorithm for Compatible Structures

Let s=s_1,s_2, ...,s_n be a RNA sequence, base $s_i \in \{A, U, C, G\}$, $1 \leq i \leq n$. The subsequence $s_{i,j} = s_i, \ldots, s_j$ is a segment of s, $1 \leq i \leq j \leq n$.

If s_i and s_j are complementary bases $\{AU, CG, UG\}$, then s_i and s_j may constitute a base pair (i, j). RNA secondary structure S is a set of base pairs (i, j) for s. Each base can at most take part in one base pair.

Suppose that $(i, j) \in S$ and $i < r < j$. If no $(i', j') \in S$ where $i < i' < r < j' < j$, then we say r is accessible from (i, j). The pair $(p, q) \in S$ is to be accessible from (i, j) if both p and q are accessible, $i < p < q < j$. The $u(\geq 0)$ unpaired bases and $k - 1(\geq 0)$ pairs, accessible from $(i, j) \in S$, constitute the k-loops closed by (i, j). 1-loops are called hairpins. 2-loops for $u = 0$ are called stacked pairs stems, for $p - i - 1 > 0$ and $j - q - 1 > 0$ are called internal loops, else for $p - i - 1 = 0$ or $j - q - 1 = 0$ are called bulges. k-loops ($k \geq 3$) are called multi-loops. Let $E_k(i, j)$ be the energy of k-loops closed by base pair $(i, j) \in S$, especially $E_2(i, k : l, j)$ denotes the energy of 2-loops closed by base pair (i, j) and $(k, l) \in S$, $i < k < l < j$.

Base pairs (i, j) and (k, l) are said to be compatible if they are juxtaposed (e.g. $i < j < k < l$) or nested (e.g. $i < k < l < j$). Otherwise they are called a pseudoknot (e.g. $i < k < j < l$).

RNA secondary structure is represented by Feyman diagram as Fig.1. Solid continuous level line represents an RNA sequence. Solid arcs represent base pairs, and dashed arcs represent that the relation between two nonadjacent bases is uncertain. Any pseudoknot whose representation in Feyman diagram requires crossing lines is called a non-planar pseudoknot, otherwise called a planar pseudoknot.

Let $W(i, j)$ be the minimum energy possible for $s_{i,j}$, and $V(i, j)$ be the minimum energy for $s_{i,j}$, where s_i and s_j constituting a base pair (i, j). The recursions of W and V can be expressed as (1) and (2). M denotes the score for generating

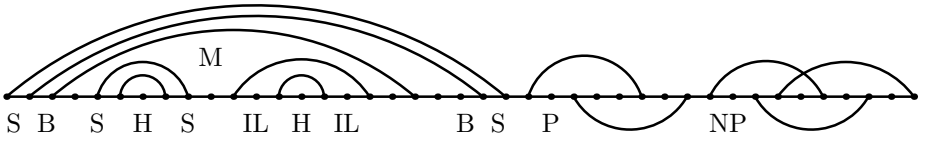


Fig. 1. The representation of RNA secondary structures including pseudoknot. H(hairpin) S(Stem) B(bulge) IL(internal loop) M(Multi-loop) P(pseudoknot) NP(non-planar pseudoknot)

a multi-loop. P denotes the score for each pair in a multi-loop. W_M has identical recursions but different parameters with W.

$$V(i,j)=min\begin{cases}E_1(i,j)\\ min\{E_2(i,j)+V(k,l)\}, i<k<l<j, u=k-i+j-l-2<U\\ min\{W_M(i+1,h)+W_M(h+1,j-1)+M+P\}, i<h<j-1\end{cases} \tag{1}$$

$$W(i,j)=min\left\{V(i,j), W(i+1,j), W(i,j-1), \min_{i<k<j-1}\{W(i,k)+W(k+1,j)\}\right. \tag{2}$$

In practice, for 2-loops, the value of u actually encountered are rarely large, say $u \leq U$, U is a constant [13]. This inherent feature is great important for the Zuker algorithm to reduce time complexity from $O(n^4)$ to $O(n^3)$. Our algorithm also applies it to reduce calculation.

3 Algorithm Including Pseudoknots

The Jens algorithm computes pseudoknots consisting of two crossed stems, and two pseudoknots can only be in nested or parallel position. In practice, the pseudoknots composed of internal loops and bulges often occur in the secondary structure. The crossed pseudoknots are also great important, and shouldn't be ignored. We design a new algorithm to predict the RNA secondary structure including the above-mentioned cases.

3.1 Basic Principle

Definition 1. *For subsequence $s_{i,j}$, if $(i,j),(i+1,j-1),\ldots,(k,l)$ are base pairs, $i \leq k < l \leq j$, then the structure closed by (i,j) and $(k,l) \in S$ is called 1-stem $S_1[i,j]$. If $S_1[i,j]$ closed by (i,j) and $(r,s) \in S, S_1[r',s']$ closed by (r',s') and $(k,l) \in S, i \leq r < r' \leq k < l \leq s' < s \leq j$ and $u = r' - r + s - s' > 2$, then the structure closed by (i,j) and $(k,l) \in S$ is called 2-stems $S_2[i,j]$. Equally if $S_1[i,j]$ closed by (i,j) and $(r,s) \in S, (k-1)$ nested 1-stems closed by (r',s') and $(k,l) \in S, i \leq r < r' \leq k < l \leq s' < s \leq j$ and $u = r' - r + s - s' > 2$, then the structure closed by (i,j) and $(k,l) \in S$ is called k-stems $S_k[i,j]$. The optimal energy of $S_k[i,j]$ is denoted as $ES_k(i,j)$. Correspondingly the lengths of optimal $S_k[i,j]$ are denoted as $LS_k(i,j) = k - i + 1$ and $RS_k(i,j) = j - l + 1$.*

Arbitrary pseudoknots can be decomposed into k-stems and multi-loops. In the PseudoBase[12], there are 244 RNA sequences including 253 pseudoknots. Pseudoknots composed of 1-stem get up to 58.5 percent, and those composed of k-stems get up to 94.5 percent, and those composed of k-stems and multi-loops get up to 100 precent of all pseudoknots.

Let $LS(i,j) \in \{LS_1(i,j), LS_2(i,j)\}, RS(i,j) \in \{LS_1(i,j), RS_2(i,j)\}, ES(i,j) \in \{ES_1(i,j), ES_2(i,j)\}$. Note that $S_2[i,j]$ is composed of two nested 1-stems and their internal unpaired bases. $ES_2(i,j) = ES_1(i,j) + E_2(r, r' : s', s) + ES_1(r',s')$.

Equally $ES_k(i,j)$ is composed of k nested 1-stem and their internal unpaired bases, $ES_k(i,j) = ES_1(i,j) + E_2(r, r' : s', s) + ES_{k-1}(r', s')$.

In our algorithm, the free energies of 1-stems and 2-stems are pre-computed and stored in triangular matrices $ES_1(i,j)$ and $ES_2(i,j)$ in $O(n^3)$ time as procedure1. Meanwhile the lengths of k-stems ($k \leq 2$) are stored in matrices $LS(i,j)$ and $RS(i,j)$.

Procedure 1: Computing the energies of 1-stem and 2-stems.

/* (i,j) denotes base pair composed of base s_i and s_j. g denotes the penalty for 2-loops in a pseudoknot. P' denotes the score for each pair in a pseudoknot. Q' denotes the score for an unpaired base in a pseudoknot.*/

```
for r=4 to n
    for i = 1 to n − r
        j = i + r; ES1(i, j) = ES2(i, j) = 0;
        if (i, j) k = i; l = j;
//Compute the energy of 1-stem
            while (k, l) and (k + 1, l − 1) and ((l − k) > 4)
                ES1(i, j) = ES1(i, j) + g ∗ E2(k, k + 1 : l − 1, l) + P′;
                k + +; l − −;
            loop
            ES1(i, j) = ES1(i, j) + P′; ES2(i, j) = ES1(i, j);
//Compute the energy of 2-stem
            if (k = i and l = j)
                for k = i + 1 to i + U + 1
                    for l = j − U − 1 + k − i to j
                        ES2(i, j)=min{ES1(i, j) + g ∗ E2(i, k : l, j)+
                                      ES1(k, l) + (k − i + j − l − 2) ∗ Q′}
                    end for
                end for
            else ES2(i, j) = ES1(i, j) + ES2(k, l);
            end if
        end if
    end for
end for
```

Definition 2. *A semi-extensible structure is defined to be two connected segments $s_{i,k}$ and $s_{l,j}$, $i < k < l < j$, satisfying (1) or (2):*

(1) There exists p and $q, i < p < q < k$, such that $s_{p,q}$ and $s_{l,j}$ constitute k-stems or multi-loops. Let $F[i,k : j]$ denote the semi-extensible structure. Correspondingly $EF(i,k : j)$ denotes the optimal energy of the semi-extensible structure. The length of the optimal semi-extensible structure is denoted as $LF(i,k : j) = j - l + 1$.

(2) There exists p and $q, l < p < q < j$, such that $s_{i,k}$ and $s_{p,q}$ constitute k-stems or multi-loops. Let $F[i : l, j]$ denote the semi-extensible structure. Correspondingly $EF(i : l, j)$ denotes the optimal energy of the semi-extensible structure. The length of the optimal semi-extensible structure is denoted as $LF(i : l, j) = k - i + 1$.

Fig.2 gives a simple pseudoknot. Dots denote bases. ( ) and [ ] denote base pairs. We can see $W(1,30) = EF(1,6:19) + EF(7:20,30) + W(13,14) = ES_1(1,19) + ES_1(7,30) + W(13,14) + W(6,6) + W(20,24)$. So one pseudoknot can be decomposed into the crossing of two semi-extensible structures and one subsequence. One semi-extensible structure can be described by k-stems. Pseudoknots can be described recursively.

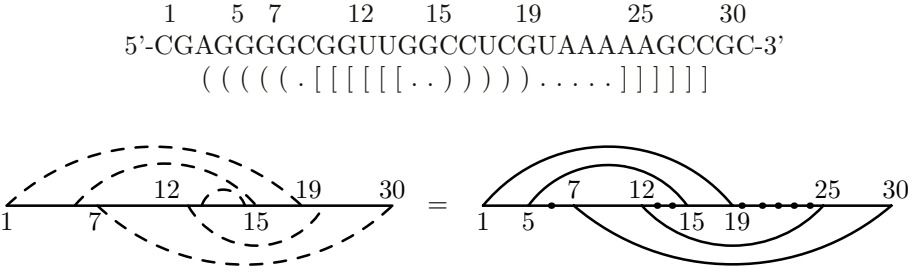


Fig. 2. Construction of a simple pseudoknot

3.2 Model and Algorithm

Applying the above principle, we extend the result of [11] to predict planar pseudoknots. Pseudoknot structures composed of two semi-extensible structures and one subsequence or one semi-extensible structure and one subsequence are added to the Zuker model to constitute our computational model. Fig.3 and Fig.4 give the illustration of our model.

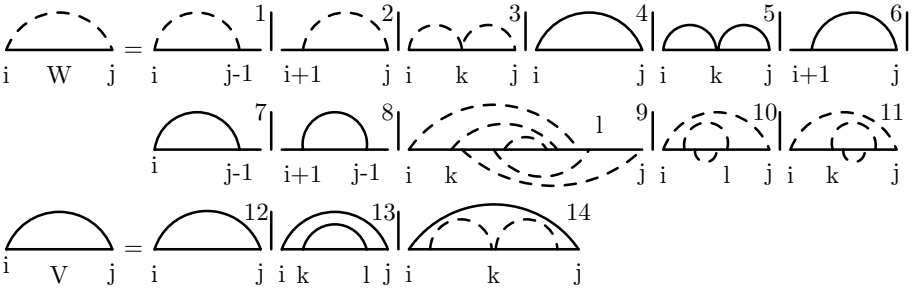


Fig. 3. The representation of W and V in the new algorithm

Fig.3 gives the computational illustration of $s_{i,j}$, which is achieved by adding Fig.3.5-3.11 to the Zuker algorithm. Fig.3.5-3.8 are used to compute the energy of coaxial stack and dangle diagrams. Fig.3.9-3.11 are used to compute the minimal free energy of pseudoknots structures.

Fig.3.9 consists of two semi-extensible structures $F[i,k:l]$ and $F[k+1:l+1,j]$, and one subsequence $s_{k+1+LF(k+1:l+1,j),l-LF(i,k:l)}$. Fig.3.10 consists of

one semi-extensible structure $F[i : l, j]$, and one subsequence $s_{i+LF(i:l,j),l-1}$. Fig.3.11 consists of one semi-extensible structure $F[i, k : j]$ and one subsequence $s_{k+1,j-LF(i,k:j)}$. The recursions of Fig.3.9-3.11 are given as (3) in our computational model. G_w denotes the score for introducing a pseudoknot.

$$\begin{array}{l} \min_{i\le k\le l\le j-1}\{EF(i,k:l)+EF(k+1:l+1,j) \\ +W_M(k+1+LF(k+1:l+1,j),l-LF(i,k:l))+G_W)\} \\ \min_{i+1\le l\le j}\{EF(i:l,j)+W_M(i+LF(i:l,j),l-1)\} \\ \min_{i\le k\le j-1}\{EF(i,k:j)+W_M(k+1,j-LF(i,k:j))\} \end{array} \tag{3}$$

Fig.3.5 indicates that if (i, k) and $(k+1, j) \in S$, this configuration is specially favored by an amount $C(i, k : k + 1, j)$, because substructures corresponding to $V(i, k)$ and $V(k + 1, j)$ are coaxial stack. Fig.3.6-3.8 indicate that one unpaired base contiguous to a base pair has a different thermodynamic contribution than other unpaired bases. The dangle scoring function $L(i : i + 1, j)$ depends both on the dangling base s_i and base pair $(i + 1, j)$. Similarly $R(j : i, j - 1)$ depends on s_j and base pair $(i, j - 1)$. The recursions of Fig.3.5-3.8 are given as (4) in our computational model.

$$\begin{array}{l} \min_{i<k<j-1}\{V(i,k)+V(k+1,j)+C(i,k:k+1,j)\} \\ L(i:i+1,j)+V(i+1,j), R(j:i,j-1)+V(i,j-1) \\ L(i:i+1,j-1)+R(j:i+1,j-1)+V(i+1,j-1) \end{array} \tag{4}$$

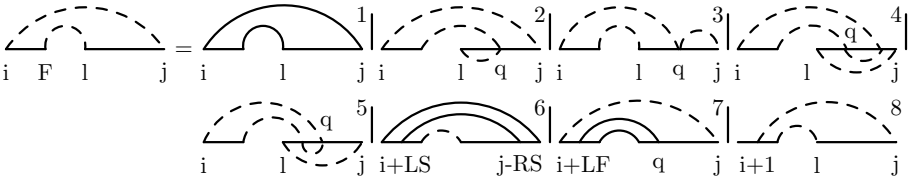


Fig. 4. The representation of F in the new algorithm

Fig.4 gives the computational illustration of semi-extensible structures $F[i : l, j]$. The computational illustration of $F[i, k : j]$ can be given correspondingly.

One semi-extensible structure is computed from five cases. One semi-extensible structure is one k-stems. For example, $F[i : l, j] = S_1[i, j]$ or $F[i : l, j] = S_2[i, j]$, if (i, j) and $(i + LS(i, j), l) \in S$ in Fig.4.1. One semi-extensible structure consists of another semi-extensible structure and one subsequence. For example, $F[i : l, j]$ consists of $F[i : q, j]$ and $s_{l,q-1}$ in Fig.4.2, $F[i : l, j]$ consists of $F[i : l, q]$ and $s_{q+1,j}$ in Fig.4.3. One semi-extensible structure consists of two other semi-extensible structures. For example, $F[i : l, j]$ consists of $F[l, q : j]$ and $F[i : q + 1, j - LF(l, q : j)]$ in Fig.4.4, $F[i : l, j]$ consists of $F[l : q, j]$ and $F[i : l + LF(l : q, j), q - 1]$ in Fig.4.5. One semi-extensible structure consists of another semi-extensible structure and one k-stems ($k \le 2$). For example, $F[i : l, j]$ consists of $S_k[i, j]$ and $F[i+LS(i, j) : l, j-RS(i, j)]$ in Fig.4.6, $F[i : l, j]$ consists of $F[i : q, j]$ and $S_k[i+LF(i : q, j), q-1]$ in Fig.4.7. One semi-extensible

structure consists of another semi-extensible structure and one unpaired base. For example, $F[i:l,j]$ consists of $F[i+1:l,j]$ and base s_i in Fig.4.8.

So the recursions of Fig.4 are given as (5) in our computational model. M' denotes the score for generating an incompatible multi-loop. G_{wh} denotes the score for overlapping pseudoknots.

$$F(i:l,j) = \min_{l<q<j} \begin{cases} ES(i,j), EF(i:q,j) + W_M(l,q-1) \\ EF(i:l,q) + W_M(q+1,j) \\ EF(l,q:j) + EF(i:q+1, j-LF(l,q:j)) + G_{wh} \\ EF(l,q:j) + EF(i:l+LF(l,q:j), q-1) + G_{wh} \\ ES(i,j) + EF(i+LS(i,j):l, j-RS(i,j)) + M' \\ ES(i+LF(i:q,j), q-1) + EF(i:q,j) + M' \\ EF(i+1:l,j) + Q' \end{cases} \tag{5}$$

Note that in Fig.4.6, if $(i+L(i,j)-1, j-RS(i,j)+1) \in S$ in $F[i+L(i,j)-1 : l, j-RS(i,j)+1]$, then the outmost layer of $F[i+L(i,j)-1:l,j-RS(i,j)+1]$ is one k-stems $S_k[i+L(i,j)-1, j-RS(i,j)+1]$. $S_k[i+L(i,j)-1, j-RS(i,j)+1]$ and $S_k[i:j]$ constitute one k-stems ($k \geq 2$). Similarly in Fig.4.7, if $(i+LF(l:q,j)-1,q) \in S$ in $F[i:q,j]$,then innermost layer of $F[i:q,j]$ and $S_k[i+LF(i:q,j)-1,q]$ constitute one k-stems ($k \geq 2$). In this case semi-extensible structure is added one k-stems ($k \leq 2$) in each computation of Fig.4.6-4.7, so k-stems ($k \geq 3$) may be constituted. Then the recursions of Fig.4.6-4.7 is described as (6).

$$\begin{array}{l} ES(i,j) + EF(i+LS(i,j)-1 : l, j-RS(i,j)+1) + M' \\ \min_{l \leq q \leq j} \{ES(i+LF(i:q,j)-1,q) + EF(i:q,j) + M'\} \end{array} \tag{6}$$

Our algorithm computes the minimum free energy of a pseudoknot based on nearest-neighbor thermodynamic model. We pre-compute the minimum free energy of 1-stem and 2-stem in procedure1, and store them in matrices ES_1 and ES_2. Then we compute the minimum free energy of the compatible structures and pseudoknot structures for each subsequence $s_{i,j}$ according to the recursions of Fig.3 and Fig.4, and store them in matrices W, V and EF. The dynamic programming algorithm works by adding one base at a time to $s_{i,j}$, and observing what the best structure is at each step. The last number to be computed, $W(1,n)$, represents the minimum free energy for the whole sequence s. Correspondingly the admissible secondary structure S is achieved by a trace back through the matrices W and V, and matrices EF, LF, ES, LS and RS.

3.3 Analysis

From procedure1, the computation of the matrices of 2-stems takes $O(n^3)$ time. From Fig.3, the computation of V takes $O(n^3)$ time, and the computation of W takes $O(n^4)$ time. From Fig.4, the computation of EF takes $O(n^4)$ time. So the algorithm requires $O(n^4)$ time.

From procedure1, pre-computed matrices take $O(n^2)$ space. The energy matrices V and W take $O(n^2)$ space. The matrices EF and LF for semi-extensible structure take $O(n^3)$ space. So the algorithm requires $O(n^3)$ space.

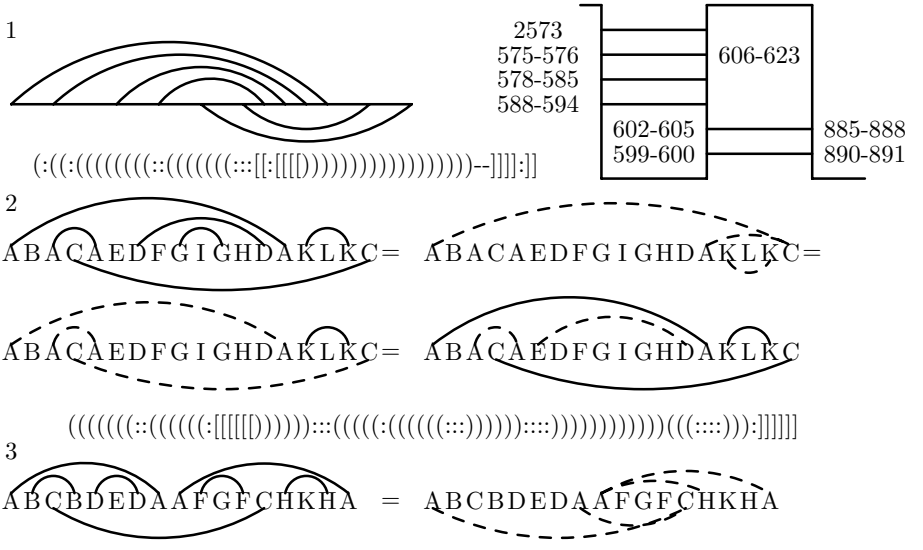


Fig. 5. The solvable pseukoknots composed of k-stems and multi-loops

The model can compute pseudoknots composed of k-stems and multi-loops such as Fig.5.1 (PKB148, Escherichia coli in [12]), Fig.5.2(PKB135, broad bean mottle virus in [12]), and Fig.5.3. For example, in Fig.5.2, $S_1[3,5]$ and $s_{6,13}$ constitute one semi-extensible structure $F[3:5,13]$ according to Fig.4.3. $F[2:5,13]$ and $S_1[1,14]$ constitute one multi-loops according to Fig.4.6. Then $F[1:5,14]$ and $F[4:18,18]$ constitute one pseudoknot.

Fig. 6. The solvable nested pseudoknots in the new algorithm

The model can also compute arbitrary nested or parallel pseudoknots. For example, in Fig.6 one pseudoknot is constituted in $s_{1,10}$ according to Fig.3.9, then $s_{1,10}$ and $s_{13,13}$ constitute one semi-extensible structure $F[1,10:13]$ according to Fig.4.2. $F[1,10:13]$ and $F[11:14,15]$ constitute another pseudoknot according to Fig.3.9, then $s_{1,10}$ contains a nested pseudoknot.

Moreover the algorithm can solve crossed pseudoknots as Fig.7.1 (PKB163, Homo sapiens in [12]) and Fig.7.2. For example, in Fig.7.2, $F[1:5,6]$ and $F[2,4:9]$ constitute another semi-extensible structure $F[1,6:9]$ containing one pseudoknot according to Fig.4.5, $F[7:12,14]$ and $F[10,11:15]$ constitute another semi-extensible structure $F[7:10,15]$ containing one pseudoknot. Then $F[1,6:9]$ and $F[7:10,15]$ constitute crossed pseudoknots.

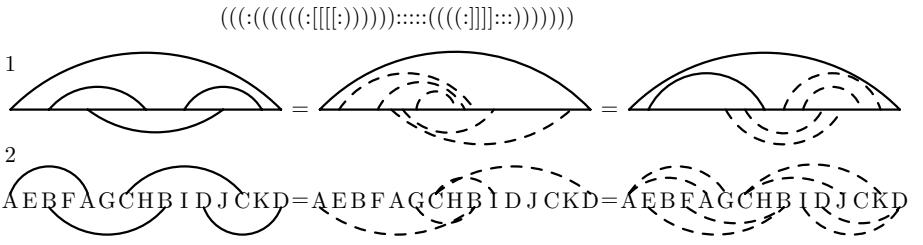


Fig. 7. The solvable crossed pseudoknots in the new algorithm

4 Discussion

We present a new dynamic programming algorithm with $O(n^4)$ time and $O(n^3)$ space to predict the optimal RNA secondary structure including pseudoknots. Semi-extensible structure and k-stems are introduced to predict the class of nested and crossed multi-pseudoknots.

Our algorithm can solve arbitrary nested or parallel pseudoknots composed of k-stems and multi-loops, and a subclass of crossed pseudoknots whose representation in Feyman diagram doesn't require crossing lines. So our algorithm can solve arbitrary planar pseudoknots.

Our algorithm can't find non-planar pseudoknots. Fig.8.1 (PKB71, Escherichia coliα mRNA in [12]) and Fig.8.2 (parallel β-sheet protein in [14]) present this structure. The Rivas algorithm can find Fig.8.1, but can't find Fig.8.2 too. It is the unique difference of our algorithm with the Rivas algorithm. So our algorithm has the same power as the Rivas algorithm for the planar pseudoknots prediction. While our algorithm is practical for sequences shorter than 1000 bases, which is better than 140 bases of the Rivas algorithm.

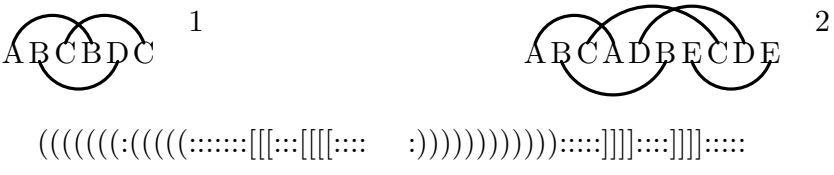


Fig. 8. The non-solvable structures in the new algorithm

This algorithm can compute 252 of 253 pseudoknots from 244 RNA sequences in the PseudoBase[12]. One non-planar pseudoknots (figure 8.1) can't be computed. The Jens algorithm can compute pseudoknots composed of two 1-stems, such as Fig.6, but can't compute pseudoknots composed of k-stems for $k > 1$ and multi-loops , such as Fig.5. The Jens algorithm can't yet compute crossed pseudoknots, such as Fig.7 and Fig.8. So our algorithm can compute more complex nested and crossed pseudoknots than the Jens algorithm.

The Lyngsϕ algorithm takes $O(n^5)$ time and $O(n^3)$ space to compute pseudoknots, where one pseudoknot consists of two compatible structures. The Lyngsϕ algorithm can compute the class of one planar pseudoknot, such as Fig.5 and Fig.7.1. The Lyngsϕ algorithm can't compute multi-pseudoknots, such as Fig.6, Fig.7.2 and Fig.8. So our algorithm can compute more complex multi-pseudoknots, and takes lower time than the Lyngsϕ algorithm.

References

1. Zuker, M. Computer prediction of RNA structure.: Methods Enzymol **180** (1989a) 262–288
2. Schuster, P., Fontana, W., Stadler, P. F. and Hofacker, I. L.: From sequences to shapes and back: a case study in RNA secondary structure. Proc. Roy. Soc. ser. B. 255 (1994) : 279–284
3. Rivas, E. and Eddy, SR.: A dynamic programming algorithm for RNA structure prediction including pseudoknots, J. Mol. Biol. **285** (1999) 2053–2068
4. Lyngsϕ, RB. and Pedersen, CN.: RNA pseudoknot prediction in energy based models. J. Mol. Biol. **7** (2001) 409–428
5. Abrahams, J. P., van der Berg, M. et al.: Prediction of RNA secondary structure, including pseudoknotting, by computer simulation. Nucleic Acids Research **18** (1990) 3035–3044
6. Gultyaev, A. P., van Batenburg, F. H. and Pleij, C. W. A.: The computer simulation of RNA folding pathways using a genetic algorithm. J. Mol. Biol. **250** (1995) 37–51
7. Akutsu, T.: Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. Discrete Applied Mathematics **104** (2000) 45–62
8. Ruan, J., Stormo, GD. and Zhang, W.: An iterated loop matching approach to the prediction of RNA secondary structures with pseudoknots. Bioinformatics **20** (2004) 58–66
9. Hengwu, Li., Daming, Zhu. and Shaohan, Ma.: An impoved algorithm for RNA secondary structure prediction including pseudoknots. TCS, Qingdao. Computer Science Press (2003) 109–111
10. Hengwu, Li., Daming, Zhu. and Mingxiao Sun.: A new algorithm for RNA secondary structure prediction. TCS, Wuhan. Computer Science Press (2004) 142–144
11. Jens, R. and Robert, G.: Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. BMC Bioinformatics **5** (2004) 104
12. PseudoBase homepage: http://wwwbio. LeidenUniv.nl/$\sim$ Batenburg/PKB.htm
13. Sankoff, D.: Simultaneous solution of the RNA folding, alignment and protosequence problems. SIAM J. Appl. Math. **45** (1985) 810–825
14. Gluick, T. C. and Draper, D. E.: Thermodynamics of folding a pseudoknotted mRNA fragment. J. Mol. Biol. **241** (1999) 246–262

Rapid Homology Search with Two-Stage Extension and Daughter Seeds*

Miklós Csűrös[1] and Bin Ma[2]

[1] Department of Computer Science and Operations Research, Université de Montréal
C.P. 6128, succ. Centre-Ville, Montréal, Qué., Canada, H3C 3J7
csuros@iro.umontreal.ca
[2] Department of Computer Science, University of Western Ontario
London, Ont., Canada, N6A 5B7
bma@csd.uwo.ca

Abstract. Using a seed to rapidly "hit" possible homologies for further examination is a common practice to speed up homology search in molecular sequences. It has been shown that a collection of higher weight seeds have better sensitivity than a single lower weight seed at the same speed. However, huge memory requirements diminish the advantages of high weight seeds. This paper describes a two-stage extension method, which simulates high weight seeds with modest memory requirements. The paper also proposes the use of so-called daughter seeds, which is an extension of the previously studied vector seed idea. Daughter seeds, especially when combined with the two-stage extension, provide the flexibility to maximize the independence between the seeds, which is a well-known criterion for maximizing sensitivity. Some other practical techniques to reduce memory usage are also discussed in the paper.

1 Introduction

An important task in the analysis of molecular sequences is the search for *local alignments*, formed by pairs of substrings from two sequences, which score high according to some string similarity metric. Local alignments are the "unit operations" in comparative genomics [1], where sequence conservation and lack of it are used to reason about evolutionary relationships and biological function. For instance, even alignments between different species' genomes [2] rely on anchors, which are local alignments between the genomes that restrict the search space for whole-genome alignments.

The importance of the local alignment problem led to a large body of research, starting in the early 1980s with the algorithm of Smith and Waterman [3], later improved by Gotoh [4]. The Smith-Waterman-Gotoh algorithm uses dynamic programming to find all local alignments scoring above a fixed threshold in $O(|S| \cdot |T|)$ time for two sequences S and T over a finite alphabet Σ. For

* This work was supported by grants from the Natural Sciences and Engineering Research Council of Canada, the *Fonds québécois de la recherche sur la nature et les technologies*, and a Canada Research Chairs program.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 104–114, 2005.

genomic sequences, Σ is the DNA alphabet of size 4. While the speed of a full sensitivity search can be improved by a logarithmic factor [5], a full-scale search that involves sequences with several million letters cannot be carried out in a reasonable time frame. For large alignment problems, other solutions are needed that may sacrifice some sensitivity for speed, i.e., that may miss some local alignments but run reasonably fast. Heuristic search programs such as FASTA [6] and BLAST [7] were introduced at the end of the 1980s. They rely on the so-called *hit-and-extend* heuristic, which can be implemented using hashing and lookup tables. The majority of modern local alignment programs [8–12] exploit some variant of this idea. Some recent alternatives are based on suffix trees [13].

Algorithm HIT-AND-EXTEND
Input sequences S, T; hash function h
H1 **for** $i = 1, \ldots, |S| - \ell + 1$ **do**
H2 set $g \leftarrow h(S[i..i + \ell - 1])$
H3 add i to the list $\mathsf{Occ}(g)$
H4 **for** $j = 1, \ldots, |T| - \ell - 1$ **do**
H5 set $g \leftarrow h(T[j..j + \ell - 1])$
H6 process hits $(i, j) : i \in \mathsf{Occ}(g)$

Algorithm X-DROP
Input S, T; start (i, j); allowed drop X
X1 set $s \leftarrow 0; \mathsf{max} \leftarrow 0$
X2 **while** $s > \mathsf{max} - X$, $i \leq |S|$, $j \leq |T|$ **do**
X3 **if** $S[i] = T[j]$ **then** $s \leftarrow s + 1$ **else** $s \leftarrow s - 1$
X4 **if** $s > \mathsf{max}$ **then** set $\mathsf{max} \leftarrow s$
X5 set $i \leftarrow i + 1; j \leftarrow j + 1$
X6 report max

Fig. 1. Basic hit-and-extend procedure. Algorithm HIT-AND-EXTEND outlines the method. Hits are extended in Line H6 by exploring the dynamic programming table around the hits. X-DROP is a popular extension algorithm, used in BLAST [7, 8] and many other alignment programs. The extension is shown only in the forward direction. An analogous extension process is carried out in the reverse direction.

This paper concentrates on hit-and-extend methods. Hit-and-extend methods rely on a hash function $h\colon \Sigma^\ell \to \{0, \ldots, H - 1\}$. Local alignments are found by first identifying *hits*, which are pairs of positions (i, j) where $h(S[i..i + \ell - 1]) = h(T[j..j + \ell - 1])$. The most obvious choice for hashing is to use the identity function, when hits are defined by identical substrings of length ℓ, called ℓ-*mers*. In fact, this strategy is used by BLAST. All the hits can be found efficiently by using a lookup table that stores the *occurrence lists* $\mathsf{Occ}(g) = \{i\colon h(S[i..i + \ell - 1]) = g\}$ for every key g. Subsequently, hits are detected and extended by consulting the occurrence list for $h(T[j..j+\ell-1])$ in each position j. Figure 1 outlines this concept. This strategy is often called "seeding" and the hash function or its representation is called as a seed. The sensitivity of a seed measures its ability to hit a homology, and the specificity of a seed characterizes its ability to filter out a random region.

It was recently discovered [11] that *spaced seeds* provide very good sensitivity and specificity. A spaced seed is defined by a set $\mathcal{S} = \{s_1, \ldots, s_k\} \subseteq \{1, \ldots, \ell\}$. In practice, a spaced seed is often denoted by the characteristic vector for the set, defined as the length-ℓ binary string in which the bits at the positions specified by the seed have value 1. The corresponding hash function concatenates the characters in positions specified by the seed, and encodes the resulting string $u[s_1] \cdot u[s_2] \cdots u[s_k]$ by an integer in the range $\{0, \ldots, |\Sigma|^k - 1\}$. Such a

seed is called an (ℓ, k)-seed, and has *weight* k. The initial discovery led to a number of results on selecting spaced seeds [14, 15] in various statistical or empirical alignment models. Additional references with a thorough discussion are offered in [16].

There exist several generalizations of spaced seeds, which include multiple seeds [12, 17], and vector seeds [9, 18]. Multiple seeds are a set of carefully selected spaced seeds $\mathcal{S}_1, \ldots, \mathcal{S}_M$. The set of hits for such a set is the union of the hits found by every single seed. A vector seed is defined by a vector of non-negative weights $(w_1, \ldots, w_\ell)$ and a threshold t: there is a hit at (i, j) if $t \le \sum_{\delta=1}^{\ell} w_\delta I\{S[i+\delta-1] = T[j+\delta-1]\}$, where $I\{C\}$ is 1 if and only if condition C is true, otherwise it is 0. (The slightly more general definition of [18] allows for a scoring matrix.) A vector seed can be viewed as a well-structured set of multiple seeds.

The time complexity increase of using multiple seeds can be offset by using higher-weighted seeds. It was shown that higher-weighted multiple seeds and vector seeds may offer superior sensitivity [12, 18] to that of a single seed at the same specificity. However, they can hardly reach their theoretical potential due to their memory requirements. In case of multiple seeds [12], a lookup table is constructed for every seed. Vector seeds rely on a hash table for the spaced seed defined by the positions with non-zero weights. As a consequence, memory usage is exponential in the number of positive weights. Vector seeds with widely varying weighting schemes proved prohibitive due to their demands on memory.

We first propose in Section 2 a novel two-stage extension procedure that improves the efficiency of hit-and-extend methods. Rather than being a trivial heuristic, extensive optimization is needed to maximize the sensitivity of the two-stage extension. The concept of daughter seeds is introduced in Section 3. Daughter seeds allow us to attain or surpass the sensitivity and speed of multiple and vector seeds, and pose only modest demands on memory. We discuss the advantages of combining two-stage extension and the daughter seeds. Section 4 explores some practical techniques of space reduction, which include an implementation of 11-mer based hashing with 1.5 bytes per base pair for the purposes of comparing mammalian-sized genomes. Section 5 concludes the paper.

2 Two-Stage Extension

2.1 Average Complexity of the Classic Hit-and-Extend Method

In this study, we restrict our attention to gapless local alignments. The presented techniques are, however, relevant also for gapped alignments, as most programs perform gapped extension only if a high-scoring gapless alignment is found. For simplicity, we consider the alignment scoring policy that rewards an identity with $+1$ and penalizes a mismatch with -1. Thus, without loss of generality, each local alignment between $S[i..i+n-1]$ and $T[j..j+n-1]$ can be represented by a 0-1 string R of length n, where $R[k] = 1$ if and only if $S[i+k-1]$ matches $T[i+k-1]$. Let n' be the number of ones in R. Then the score of the alignment is $(2n' - n)$. The *similarity* of the local alignment is then ratio n'/n.

If $S[i..i+n-1]$ and $T[j..j+n-1]$ are random unrelated sequences, then the similarity is expected to be $\beta = \sum_{a \in \Sigma} p(a)q(a)$, where $p(a)$ and $q(a)$ are the background frequencies for the letter a in the two sequences. For DNA sequences with alphabet size 4, $\beta = \frac{1}{4}$ if the letter occurrences are uniform random in at least one of S and T. For simplicity, we mostly focus on such a model of random sequences. Nonetheless, the analyses can be easily extended to arbitrary background frequencies.

A heuristic local alignment method can be assessed by evaluating its specificity and sensitivity. *Specificity* is measured by the average running time on random unrelated sequences. *Sensitivity* is measured by the probability of detecting a homology under a probabilistic model of homologies.

Since the introduction of spaced seeds, there has been much work on finding variants of hit conditions and hash functions to gain better sensitivity and specificity. In this section we scrutinize the extension instead. The usual method is to extend a hit in each of the two directions along the diagonal until the score drops by a specified amount. In each direction, the position where the maximum score is reached is recorded and gives the extent of the local alignment. Figure 1 shows the X-DROP extension procedure in one direction.

If we ignore the boundary effects of S and T, the average running time of a hit-and-extend method for random sequences is $f \times t \times |S| \times |T|$, where f is the probability of a hit at a fixed position pair (i, j), and t is the average time spent on a hit extension. The probability f is called the *false positive rate* in [18]. In what follows, we analyze t more closely for the X-drop algorithm of Fig. 1. In Line X3, the score decreases with an expected value of $(1 - 2\beta)$ in each step. Therefore the extension will stop after around $X/(1 - 2\beta)$ steps. The following lemma formalizes this argument.

Lemma 1. *Suppose that $\beta < 1/2$ holds for the match probability, and* X-DROP *is invoked with a positive integer X. Let $n = \min\{|S|, |T|\}$. If τ denotes the number of times the loop body X3–X5 is executed, then*

$$\lim_{n \to \infty} \mathbb{E}\tau = \frac{X - \beta\Big(1 - \big(\frac{\beta}{1-\beta}\big)^X\Big)}{1 - 2\beta} = \frac{X - \sum_{t=1}^{X} \big(\frac{\beta}{1-\beta}\big)^t}{1 - 2\beta}. \tag{1}$$

Proof. Omitted due to space constraints. The proof relies on an analysis of ladder points which are the places where max is updated in Line X4.

Corollary 1. *For uniform random DNA sequences, $\beta = \frac{1}{4}$, and the expected number of comparisons at a hit is $4X - 2 + o(1)$.*

With a typical choice $X = 16$, a hit extension entails approximately 62 character comparisons on average.

2.2 Two-Stage Hit Extension

We propose the following two-stage extension process. Let $\mathcal{S} = \{s_1, \ldots, s_k\}$ be an (ℓ, k)-seed, and let $\mathcal{S}' = \{s'_1, \ldots, s'_m\}$ be a set of positive integers with $\mathcal{S} \cap \mathcal{S}' = \emptyset$.

Furthermore, let $0 < t \leq m$ be a threshold. The triple $(\mathcal{S}, \mathcal{S}', t)$ defines a *relaxed seed* employed in the following manner. Hits are found as if the spaced seed $\mathcal{S}$ were used. When a hit is found, the positions of $\mathcal{S}'$ are tested, and the full extension is performed if at least t matches are found. In particular, let (i, j) be a hit position. Full extension is performed only if $S[i + s' - 1] = T[j + s' - 1]$ for at least t of $s' \in \mathcal{S}'$. A relaxed seed may significantly increase the specificity, which can be seen in Theorem 1. As we will see in Table 1, the sensitivity of a relaxed seed varies very much for different choices of $\mathcal{S}'$ even with the same size and threshold. Therefore, the optimization of the positions in $\mathcal{S}'$ should be done together with $\mathcal{S}$.

Theorem 1. *Suppose that the two-stage extension method is employed with* $|\mathcal{S}'| = m$. *Let* $b(m,t) = \sum_{i=t}^{m} \binom{m}{i} (\frac{1}{4})^i (\frac{3}{4})^{m-i}$. *The average number of character comparisons performed during a bi-directional hit extension is* $C = \big(m + b(m,t)(4X - 2)\big) + o(1)$.

Proof. (Omitted; follows from Lemma 1)

Sensitivity is assessed in the following manner. Let R be a binary representation of a homology region with a given similarity. In order to have a hit with the spaced seed $\mathcal{S}$, R has to contain a substring u such that $\forall s \in \mathcal{S}\colon u[s] = 1$. In order to have a hit with the relaxed seed $(\mathcal{S}, \mathcal{S}', t)$, R has to contain a substring u such that $\forall s \in \mathcal{S}\colon u[s] = 1$ and $\sum_{j=1}^{m} u[s'_j] \geq t$. The sensitivity is defined to be the hit probability under a specific probabilistic model for homologies. The first such model was introduced in [11]: it imposes that local alignments are created by independent equiprobable substitutions. Here we consider similarity patterns drawn uniformly from the set of length-n binary strings in which there is a 1 in exactly k positions. Computing the sensitivity of spaced seeds in a similar model was considered by Kucherov et al. [19]. Theirs and earlier algorithms for computing the sensitivity of spaced seeds [14, 15] can be readily adapted to relaxed seeds. As an alternative, one can convert a relaxed seed to an equivalent set of multiple seeds and compute the sensitivity by employing the algorithm in [12] that calculates the sensitivity of an arbitrary set of seeds.

Table 1 compares relaxed and spaced seeds. It turns out that the sensitivity of a $(k-1)$-weight seed can be approached while the running time stays close to that of a weight-k spaced seed. It is noteworthy that the last two seeds, `x1110x10x10x1010111` and `1110010010010101111xxxx` have the same $\mathcal{S}$, same size $|\mathcal{S}'|$ and same threshold t. At the same time, they have very different sensitivities. The example demonstrates that the two-stage extension is not a trivial extension heuristic. Indeed, it can be fully profited of only after a meticulous optimization step, in which the threshold and the relaxed positions are selected. This observation is epitomized by the extreme case of a relaxed seed $(\mathcal{S}, \mathcal{S}', t)$ where $t = |\mathcal{S}'|$. This relaxed seed is equivalent to the spaced seed $\mathcal{S} \cup \mathcal{S}'$, and the necessity to optimize the spaced seed is well-known [11].

The data structure for the basic algorithm of Fig. 1 has to support the operation $\textsc{Add}(g, i)$ that records the position i as one belonging to $\mathsf{Occ}(g)$, and the operation $\textsc{reportAll}(g)$ that returns the list $\mathsf{Occ}(g)$ as a set. For a spaced

Table 1. Comparison of some relaxed and spaced seeds. Relaxed seeds are encoded by `0`–`1`–`x` strings: position i has `1` if it is in $\mathcal{S}$, and it has `x` if it is in $\mathcal{S}'$. Sensitivity (*Sens.*) is calculated at a 70% similarity level for homology regions of lengths 64. Column C shows the expected number of comparisons in hit extension, when $X = 16$, and column T lists the expected time spent on finding and extending hits, defined as C times the false positive rate. T is normalized for weight-11 seeds. The two spaced seeds on the left are the most sensitive weight-10 and weight-11 seeds. The table on the right-hand side lists some relaxed seeds. It illustrates that the placement of relaxed positions has a non-negligible effect on sensitivity.

Seed	Sens.	C	T
111001001001010111	0.618	62	4
111010010100110111	0.451	62	1
1111111111	0.391	62	4

Seed & threshold		Sens.	C	T
111xx1xx1x01010111x	3	0.555	14.50	0.94
x1110x10x10x1010111	2	0.550	20.22	1.30
11100100100101011xxxx	2	0.528	20.22	1.30

seed with weight k, a rather straightforward implementation was introduced in [11]. An integer array, *head*, of length 4^k was used to record the first occurrence of each hash value. Then another integer array, *next*, of length $|S|$ is used to retrieve all the other occurrences. $next[i]$ records the next occurrence of the same hash value as position i. The two arrays *head* and *next* form a hash table that requires memory for $(4^k+|S|)$ integers. In a direct manner, a relaxed seed $(\mathcal{S}, \mathcal{S}', t)$ can be implemented by relying on the hash table for the spaced seed $\mathcal{S}$.

3 Daughter Seeds

The vector seed idea [18] is very effective for improving sensitivity. Every vector seed corresponds to a particular set of ordinary spaced seeds defined as follows. Let $(w_1, \ldots, w_\ell)$ be the weights and t be the threshold of the vector seed. Let $\mathcal{P} = \{\delta\colon w_\delta > 0\}$ be the set of positively weighted positions, and $K = |\mathcal{P}|$. The vector seed is equivalent to the set of seeds $\{\mathcal{S}_1, \ldots, \mathcal{S}_M\}$ where $\mathcal{S}_i$ are the subsets of $\mathcal{P}$ in which $t \le \sum_{\delta\in\mathcal{S}_i} w_\delta$. For convenience, we call $\mathcal{P}$ the *parent seed*, and the multiple seeds produced from the parent seed are called *daughter seeds*. When all vector weights are 0 or 1, daughter seeds are generated from the parent by removing up to $(K - t)$ elements from the parent's set of sampled positions. In fact, an equivalent set can be created by removing exactly that many positions.

Our relaxed seeds also define sets of daughter seeds: a relaxed seed $(\mathcal{S}, \mathcal{S}', t)$ is equivalent to the family of $(k + t)$-element subsets of the parent seed $\mathcal{S} \cup \mathcal{S}'$, in which only t elements of $\mathcal{S}'$ are present. The vector seed is different from the multiple seeds introduced in [12], which selects seeds from the complete seed space by a greedy algorithm. The advantage of multiple seeds over the vector seed is that the selected seeds are not dependent on each other. As a result, more local alignments may be hit by a constant number of seeds. On the other hand, multiple seeds require one hash table for each seed, which increases the memory requirements, and therefore only a few number of seeds can be used in practice. The vector seed only requires one hash table for the parent seed. As pointed out in [18], memory will still be a problem when there are more than 14 positive weights.

In what follows, we examine daughter seed sets without constraints on which positions may vary, otherwise imposed by vector seeds and relaxed seeds. We show that daughter seeds can have superior sensitivity to those of spaced seeds and vector seeds with comparable specificity, while using a reasonable amount of memory (one hash table for the parent seed).

The problem of selecting an optimal set of daughter seeds is likely to be intractable, based on NP-hardness results of selecting multiple seeds [12] or even one optimal seed (M. Li and B. Ma, manuscript in preparation). Computing the false positive rate involves some further complications. Let $\mathcal{S}_1, \ldots, \mathcal{S}_M \subseteq \mathcal{P}$ be a set of daughter seeds. In order to obtain the false positive rate, one needs to count subsets of $\{1, \ldots, \ell\}$ that are supersets of at least one of the seeds, where ℓ is the common seed length (i.e., $\ell = \max \cup_{i=1}^{M} \mathcal{S}_i$). Since neither the sensitivity nor the specificity changes when we add a new daughter seed $\mathcal{D} \supset \mathcal{S}_i$, we can safely assume that the daughter set is *complete* in the sense that if $\mathcal{D} \subseteq \mathcal{P}$ is included, then so are all its supersets $\mathcal{D}'$ with $\mathcal{D} \subseteq \mathcal{D}' \subseteq \mathcal{P}$.

In order to select complete daughter sets, we used the same greedy algorithm as in Li *et al.* [12], adding daughters one by one. Let f be the false positive rate of the parent seed and $K = |\mathcal{P}|$. Suppose now that one daughter seed $\mathcal{S}_1$ is selected by removing one element of $\mathcal{P}$. Obviously, the false positive rate of that single daughter seed increases the false positive rate by $3f$. By adding another daughter seed $\mathcal{S}_2$ of the same weight, the total false positive rate becomes $7f$. Now, consider the choice between adding additional three $(K-1)$-size daughter seeds, or the $(K-2)$-daughter seed $\mathcal{S}_1 \cap \mathcal{S}_2$. Both choices increase the false positive rate by $9f$, and thus the question is which one increases the sensitivity more. Intuitively, adding three $(K-1)$-sized daughter seeds is a better choice, and we observed that behavior in experiments. We thus considered selecting complete daughter sets by first including the parent, then daughter seeds of weight $(K-1)$ until all of them are included, then daughter seeds of weight $(K-2)$ until all of them are included, and so on, down to weight-K' daughters among which not all are selected necessarily. In practice, good daughter seed sets are found with $K = 13$ or $K = 14$, and $K' \geq K-2$, and thus selecting a quasi-optimal set is feasible. If the number of weight K' daughters is M', then the false positive rate of such a set is $\left(3^{K-K'}M' + \sum_{i=0}^{K-K'+1} \binom{K}{i} 3^i\right) f$. Table 2 shows some daughter seeds. The table illustrates that daughter seeds have better sensitivity than spaced seeds, or practically implementable vector seeds with comparable false positive rates.

The idea of daughter seeds and the two-stage extension can be combined together to further improve the sensitivity and specificity. Because the two-stage extension of different daughter seeds can be done separately, we can choose different checkpoints, $\mathcal{S}'$, for different daughter seeds. This resembles the multiple seeds idea and gives the flexibility to minimize the dependency between different daughter seeds. Therefore, the sensitivity can be maximized. For instance, a 17-element set of weight-11 daughters of a weight-13 parent used in conjunction with two-hit extension has a sensitivity of 0.917 at the same speed as MD-25-14, which is faster than a weight-10 spaced seed. More results about the combination will be included in the full version of this extended abstract.

Table 2. Daughter seeds. Sensitivity values are given for length-64 regions at 70% similarity level. In case of MD seeds (sensitivity marked with *), the values are calculated from simulations involving one million random similarity regions: the accuracy is thus within ±0.002 with probability 99.9%. The "Daughters" column describes a minimal daughter set (the largest antichain) selected from the complete daughter set: MD-11-13 for instance is a set of 8 weight-12 daughters and 3 weight-11 daughters. The weight-13 and weight-14 parents are spaced seeds with maximum sensitivity among spaced seeds with equal weight. False positive rate is normalized by that of a weight-11 spaced seed. The VS seeds are vector seeds from [18].

Name	Parent weight	Daughters	Sensitivity		False positive rate
MD-3-13	13	$3 \times \text{wt}12$	0.473	*	0.625
MD-5-13	13	$5 \times \text{wt}12$	0.593	*	1.0
MD-11-14	14	$2 \times \text{wt}12 + 9 \times \text{wt}13$	0.729	*	0.95
VS-12-13	13	$13 \times \text{wt}12$	0.835		2.5
MD-16-14	14	$13 \times \text{wt}12 + 3 \times \text{wt}13$	0.841	*	2.5
MD-11-13	13	$3 \times \text{wt}11 + 8 \times \text{wt}12$	0.884	*	4.19
MD-25-14	14	$23 \times \text{wt}12 + 2 \times \text{wt}13$	0.902	*	3.91
VS-11-12	12	$12 \times \text{wt}11$	0.927		9.25
MD-14-13	13	$12 \times \text{wt}11 + 2 \times \text{wt}12$	0.941	*	9.25

4 Two Ideas for Reducing Memory Usage

Daughter seeds rely on a single hash table for the parent seed, and avoid this way the impractical memory requirements of general seed sets. Further memory reductions can be achieved by storing the hash table in a more compact fashion. We need a data structure that supports the operation REPORTALL. If k-mers are used, then a suffix array can provide the functionality, which can be implemented using $O(|S|)$ bits [20] in addition to storing the sequence S. Various self-indexing methods [21, 22] promise even better compression by storing S and the indexing structure together. These latter, however, are still impractical for genomic DNA sequence comparisons, since the amount of time they spend on retrieving each hit is measured in milliseconds [22]. Given that the number of hits between two sequences of length 10^8 is about $2.4 \cdot 10^9$ (when using a 11-weight key), the implied running time (in the order of several weeks) is unacceptable. When changing the data structure, even a four-fold increase in the execution of REPORTALL is undesirable, since in a conventional implementation shorter hash keys imply the same increase in running time, with the added benefits of reduced memory usage and improved sensitivity.

The data structure for the hashing is typically implemented using 32-bit integers [11]. Consequently, a table for a k-weight key occupies $4(4^k + |S|)$ bytes. We describe a way of saving space without much sacrifice in either speed or ease of implementation. In particular, we show how to replace 32-bit integers with $(2k)$-bit integers. For seeds of weights 10–13, this means a memory reduction of 37.5–18.75%. The idea is fairly simple: choose a large integer Q and store the modulo Q remainders in both head and next. The integer value Q is reserved for marking ends of lists, so $\lceil \log_2(Q+1) \rceil$-bit integers suffice. Figure 2 shows the

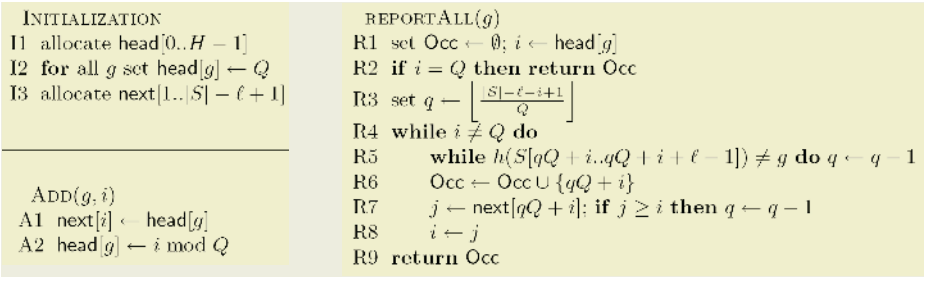


Fig. 2. Data structure for occurrence lists that uses integers in the range $\{0, \ldots, Q\}$. The value Q represents a null pointer.

data structure. Since ADD(g, i) is called in increasing order of i (cf. Fig. 1), key occurrences are restored correctly.

Theorem 2. *(a)* REPORTALL *of Fig. 2 correctly enumerates the occurrences of a key g, provided that the calls* ADD(g, i) *were made in increasing order of i.*
(b) Suppose that S is a uniform random string, and the hash function is such that all keys occur with equal probability. If $Q \gg 1$ and $|S| \to \infty$, then the hash function is evaluated in the loop of Line R5 $(1 - e^{-Q/H})^{-1}$ times on average. If h is defined by a weight-k spaced seed, then, for each occurrence of a key g, REPORTALL(g) *performs an expected number of $k + \frac{4/3}{e^{Q/H}-1}$ character comparisons.*

Proof. (Omitted due to space constraints. The proof relies on a Poisson process approximation of key occurrences.)

By Theorem 2, using $Q = 4^k - 1$ with a weight-k seed entails an expected number of $(k + 0.77)$ character comparisons. As an alternative to the (mod$Q$) representation, one can avoid the character comparisons by using run-length encoding [23] of the distances between consecutive occurrences, which reduces the space equivalently at the price of having to handle bit vectors of varying length.

Suffix trees or arrays can be employed to enumerate occurrences of k-mers. To our knowledge, there is no efficient way of retrieving occurrences of spaced seeds from a suffix array, and thus their use is limited to k-mers. At the same time, suffix tree-based local alignment methods use at least 12.5–15.6 bytes [13] per base pair. Here we describe a simple method of reducing storage for hashing with k-mers in genome-size local alignments. The idea is to use a hash table for longer $(k + d)$-mers sampled in every $(d + 1)$-th position of S. The occurrences of a key g can be retrieved by listing the occurrences of the keys $a_1 a_2 \cdots a_d \cdot g$, $a_1 \cdots a_{d-1} g a_d, \ldots, g a_1 a_2 \cdots a_d$ for all choices of $a_1, \ldots, a_d \in \Sigma$. With a judicious choice of d, the running time remains essentially the same, while the memory usage is reduced. Table 3 shows some numerical values, for a typical mammalian chromosome or genome. For instance, about 1.5 bytes/nucleotide suffice for 11-mer based alignment of a whole mammalian genome, if the sequence is stored

Table 3. Number of bits used per character when storing a k-mer table. The traditional implementation uses 32-bit integers; the implementation of Fig. 2 uses $2k$-bit integers. Sequence lengths are $|S| = 2^{27}$ for a chromosome, and $|S| = 2^{31}$ for a genome, based on the human genome.

table	chromosome		genome	
	int32	mod$Q$	int32	mod$Q$
11mers	33	22.69	32.07	22.04
every 2^{nd} 12mer	20	14.375	16.25	11.68
every 4^{th} 14mer	136	110.5	12	9.75

table	chromosome		genome	
	int32	mod$Q$	int32	mod$Q$
12mers	36	27	32.5	24.38
every 2^{nd} 13mer	32	25	17	13.28

in 2 bits/nucleotide and the table is stored in less than 10 bits/nucleotide. This memory usage is better than that of the currently most space-efficient suffix array representation [24], which uses 12 bits per nucleotide in addition to the sequence storage. At the same time, the hash table takes considerably less effort to implement.

5 Conclusion

We introduced novel ideas on selecting a structured set of spaced seeds to gain superior sensitivity and speed in hit-and-extend methods of local alignment. Our guideline in designing the techniques was to minimize memory usage, in order to avoid the main obstacle encountered by other methods such as multiple seeds and vector seeds. We described some additional, easily implementable ways to lower memory demands. Memory usage is a key factor in the efficiency of homology search algorithms, and is likely to become even more important in the future. Both the number and total length of DNA sequences in Genbank has doubled about every 17 months since 1983. This rate of increase is comparable to the popular version of Moore's law about computing power doubling every 18 months, and thus powerful heuristics are likely to remain highly valued in the comparison of molecular sequences. Our methods are memory efficient and offer practical solutions for the alignment of large genomic sequences in terms of speed and sensitivity.

References

1. Miller, W., Makova, K.D., Nekrutenko, A., Hardison, R.C.: Comparative genomics. Annu. Rev. Genomics Hum. Genet. **5** (2004) 15–56
2. Frazer, K.A., Elnitski, L., Church, D.M., Dubchak, I., Hardison, R.C.: Cross-species sequence comparisons. Genome Res. **13** (2003) 1–12
3. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. J. Mol. Biol. **147** (1981) 195–197

4. Gotoh, O.: An improved algorithm for matching biological sequences. J. Mol. Biol. **162** (1982) 708–708
5. Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A sub-quadratic sequence alignment algorithm for unrestricted cost matrices. In: Proc. SODA (2002) 679–688
6. Pearson, W.R., Lipman, D.J.: Improved tools for biological sequence comparison. Proc. Natl. Acad. Sci. USA **85** (1988) 2444–2448
7. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Mol. Biol. **215** (1990) 403–410
8. Altschul, S.F., et al: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. **25** (1997) 3389–3402
9. Schwartz, S., et al: Human-mouse alignments with BLASTZ. Genome Res. **13** (2003) 103–107
10. Ning, Z., Cox, A.J., Mullikin, J.C.: SSAHA: A fast search method for large DNA databases. Genome Res. **11** (2001) 1725–1729
11. Ma, B., Tromp, J., Li, M.: PatternHunter: faster and more sensitive homology search. Bioinformatics **18** (2002) 440–445
12. Li, M., Ma, B., Kisman, D., Tromp, J.: PatternHunter II: highly sensitive and fast homology search. J. Bioinform. Comput. Biol. **2** (2004) 411–439
13. Kurtz, S., et al: Versatile and open software for comparing large genomes. Genome Biol. **5** (2004) R12
14. Buhler, J., Keich, U., Sun, Y.: Designing seeds for similarity search in genomic DNA. J. Comp. Syst. Sci. **70** (2005) 342–363
15. Keich, U., Li, M., Ma, B., Tromp, J.: On spaced seeds for similarity search. Discrete Appl. Math. **138** (2004) 253–263
16. Brown, D.G., Li, M., Ma, B.: A tutorial of recent developments in the seeding of local alignment. J. Bioinform. Comput. Biol. **2** (2004) 819–842
17. Sun, Y., Buhler, J.: Designing multiple simultaneous seeds for DNA similarity search. In: Proc. RECOMB (2004) 76–84
18. Brejová, B., Brown, D., Vinař, T.: Vector seeds: An extension to spaced seeds. J. Comp. Syst. Sci. **70** (2005) 364–380
19. Kucherov, G., Noé, L., Ponty, Y.: Estimating seed sensitivity on homogeneous alignments. In: Proc. BIBE (2004) 387–394
20. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In: Proc. STOC (2000) 397–406
21. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: Proc. FOCS (2000) 390–398
22. Mäkinen, V., Navarro, G.: Compressed compact suffix arrays. In: Proc. CPM (2004), LNCS 3109, 421–433
23. Golomb, S.W.: Run-length encodings. IEEE Trans. Inform. Theory **12** (1966) 399–401
24. Hon, W.K., Sadakane, K.: Space-economical algorithms for finding maximal unique matches. In: Proc. CPM (2002), LNCS 2373, 144–152

On the Approximation of Computing Evolutionary Trees

Vincent Berry*, Sylvain Guillemot, François Nicolas, and Christophe Paul

Département Informatique, L.I.R.M.M. - C.N.R.S.
161 rue Ada, 34392 Montpellier Cedex 5
{vberry,sguillem,nicolas,paul}@lirmm.fr

Abstract. Given a set of leaf-labelled trees with identical leaf sets, the well-known MAST problem consists of finding a subtree homeomorphically included in all input trees and with the largest number of leaves. MAST and its variant called MCT are of particular interest in computational biology. This paper presents positive and negative results on the approximation of MAST, MCT and their complement versions, denoted CMAST and CMCT.
For CMAST and CMCT on rooted trees we give 3-approximation algorithms achieving significantly lower running times than those previously known. In particular, the algorithm for CMAST runs in linear time. The approximation threshold for CMAST, resp. CMCT, is shown to be the same whenever collections of rooted trees or of unrooted trees are considered. Moreover, hardness of approximation results are stated for CMAST, CMCT and MCT on small number of trees, and for MCT on unbounded number of trees.

1 Introduction

Given a set of leaf-labelled trees with identical leaf sets, the well-known Maximum Agreement SubTree problem (MAST) consists of finding a subtree homeomorphically included in all input trees and with the largest number of leaves [2, 7, 10, 13, 21, 22]. In other words, this involves selecting a largest set of input leaves such that the input trees are isomorphic, i.e. *agree* with each other, when restricted to these leaves.

This problem arises in various areas including phylogenetics which is concerned with *evolutionary trees*, i.e. trees representing the evolutionary history of a set of species: the leaves of the tree are in one-to-one correspondence with species under study and the branching pattern of the tree describes the way in which speciation events lead from ancestral species to more recent ones. In phylogenetics, the MAST problem is used to reach different practical goals: to obtain a consensus of several trees inferred by different methods, or that are optimal for a given criteria; to measure the similarity between different evolutionary scenarii; to identify horizontal transfers of genes. Recently, MAST has

* Supported by the *Act. Incit. Inf.-Math.-Phys. en Biol. Mol.* [ACI IMP-Bio] and the *Act. Inter. Incit. Région.* [BIOSTIC-LR].

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 115–125, 2005.

been extended to the context of supertrees where input trees can have different sets of leaves [3].

The Maximum Compatible Tree problem (MCT) is a variant of MAST that is of particular interest in phylogenetics when the input trees are not binary [11, 12, 15, 17]. MCT requires that selected subtrees of the input trees are *compatible*, *i.e.* that groups of leaves they define can all be combined in a same tree. This is less strict than requiring the isomorphism of the subtrees, hence usually leads to selecting a larger set of leaves than allowed by MAST.

We give below a brief overview of the litterature, precising how the results presented in this paper relate to previously known results. The MAST problem is NP-hard on three rooted trees of unbounded degree [2], and MCT on two rooted trees if one of them is of unbounded degree [17]. Subquadratic algorithms have been proposed for MAST on two rooted n-leaf trees [7, 20, 21]. When dealing with k rooted trees, MAST can be solved in $O(n^d + kn^3)$ time provided that the degree of one of the input trees is bounded by d [2, 6, 10], and MCT can be solved in $O(2^{2kd}n^k)$ time provided that all input trees have degree bounded by d [12]. Both problems can be solved in $O(\min\{3^p kn, 2.27^p + kn^3\})$ time, i.e. are FPT in p, where p is the smallest number of leaves to be removed from the input set of leaves so that the input trees agree [3].

More generally, when the previously mentioned parameters are unbounded, several works (starting from [2]) propose 3-approximation algorithms for CMAST and CMCT, where CMAST, resp. CMCT, is the complement version of MAST, resp. MCT, i.e. aims at selecting the smallest number of leaves to be removed from the input trees in order to obtain their agreement. In practice, input trees usually agree on the position of most leaves, thus approximating CMAST and CMCT is more relevant than approximating MAST and MCT. For CMCT, [11] propose an $O(k^2n^2)$ time 3-approximation algorithm. We propose here an $O(n^2 + kn)$ time algorithm. For MAST, [3] propose an $O(kn^3)$ time algorithm. Here we improve on this result by providing a linear time, i.e. $O(kn)$, algorithm. We also state that rooted and unrooted versions of CMAST (and CMCT) have the same approximation threshold.

Let k-MAST, resp. k-MCT, resp. k-CMAST, resp. k-CMCT, denote the particular case of MAST, resp. MCT, resp. CMAST, resp. CMCT, dealing with k rooted trees. Negative results for these problems are as follows:

- For all $\epsilon > 0$, the general MAST problem is not approximable within $n^{1-\epsilon}$ unless NP = ZPP [5]. A similar result is obtained here for MCT.
- It also stated here that 3-CMAST and 2-CMCT are APX-hard, i.e. that they do not admit a PTAS unless P = NP.
- For all $\delta < 1$, 3-MAST is not approximable within $2^{\log^\delta n}$ unless NP $\subseteq$ DTIME$\left(2^{\text{polylog}\, n}\right)$ [17]. The same result is obtained here for 2-MCT.

2 Definitions and Preliminaries

A rooted *evolutionary tree* is a tree whose leaf set $L(T)$ is in bijection with a label set, and whose internal nodes have at least two children. Hereafter, we only

consider such trees and identify leaves with their respective labels. The *size* of a tree T (denoted $\#T$) is the number of its leaves: $\#T = \#L(T)$.

Let u be a node of a tree T, $S(u)$ stands for the subtree rooted at u, $L(u)$ for the leaves of this subtree, and $d^+(u)$ for the number of children of u. For a set of leaves $L \subseteq L(T)$, $lca_T(L)$ denotes the lowest common ancestor of leaves L in T. Given a set L of labels and a tree T, the *restriction* of T to L, denoted $T|L$, is the tree homeomorphic to the smallest subtree of T connecting leaves of L.

Lemma 1. *Let T_1 and T_2 be two isomorphic trees with leaf set L, and let $L' \subseteq L$, then $T_1|L'$ is isomorphic to $T_2|L'$.*

Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of trees on a same leaf set L of cardinality n, an *agreement subtree* of $\mathcal{T}$ is any tree T with leaves in L s.t. $\forall T_i \in \mathcal{T}$, $T = T_i|L(T)$. The MAST problem consists in finding an agreement subtree of $\mathcal{T}$ with the largest number of leaves. We denote $MAST(\mathcal{T})$ such a tree.

A tree T *refines* a tree T', if T' can be obtained by collapsing certain edges of T, (i.e. merging their extremities). More generally, a tree T refines a collection $\mathcal{T}$, whenever T refines all T_i's in $\mathcal{T}$. Given a collection $\mathcal{T}$ of k trees with identical leaf set L of cardinality n, a tree T with leaves in L is *compatible with* $\mathcal{T}$ iff $\forall T_i \in \mathcal{T}$, T refines $T_i|L(T)$. If there is a tree T compatible with $\mathcal{T}$ s.t. $L(T) = L$, i.e. that is a common refinement of all trees in $\mathcal{T}$, then the collection $\mathcal{T}$ is *compatible*. In this case, a *minimum refinement* T of $\mathcal{T}$ (*i.e.* collapsing a minimum number of edges) is a tree s.t. any tree T' refining $\mathcal{T}$ also refines T. Collections of trees considered in practice are usually not compatible, motivating the MCT problem which aims at finding a tree, denoted $MCT(\mathcal{T})$, compatible with $\mathcal{T}$ and having the largest number of leaves. Remark that MCT is equivalent to MAST when input trees are binary.

For any three leaves a, b, c in a tree T, there are only three possible *binary* shapes for $T|\{a, b, c\}$, denoted $a|bc$, resp. $b|ac$, resp. $c|ab$, depending on their innermost grouping of leaves (bc, resp. ac, resp. ab). These binary trees on 3 leaves are called *rooted triples*. Alternatively $T|\{a, b, c\}$ can be a *fan*, *i.e.* a unique internal node connected to the three leaves. A fan is denoted $\{a, b, c\}$.

We define $rt(T)$, resp. $f(T)$, as the set of rooted triples, resp. fans, induced by the leaves of a tree T. Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of trees with leaf set L, a set $\{a, b, c\} \subseteq L$ is a *hard conflict* between (trees of) $\mathcal{T}$ whenever $\exists T_i, T_j \in \mathcal{T}$ s.t. $a|bc \in rt(T_i)$ and $b|ac \in rt(T_j)$. The set $\{a, b, c\}$ is a *soft conflict* between (trees of) $\mathcal{T}$ whenever $a|bc \in rt(T_i)$ and $\{a, b, c\} \in f(T_j)$.

Lemma 2 ([2, 3, 12]). *Two trees with the same leaf set are isomorphic iff there is no* hard nor any soft *conflict between them. A collection $\mathcal{T}$ of trees with the same leaf set is compatible iff there is no* hard *conflict between $\mathcal{T}$.*

Definition 1. *Given a set of conflicts $\mathcal{C}$, let $L(\mathcal{C})$ denote the leaves appearing in $\mathcal{C}$. Given a collection $\mathcal{T}$ with conflicts, an* hs-peacemaker*, resp.* h-peacemaker*, of $\mathcal{T}$ is any set $\mathcal{C}$ of disjoint hard and soft, resp. only hard, conflicts between $\mathcal{T}$ s.t. $\{T_i|(L - L(\mathcal{C})) : T_i \in \mathcal{T}\}$ is a collection of isomorphic trees, resp. compatible*

trees. In other words, removing $L(\mathcal{C})$ from the input trees removes all conflicts, resp. all hard conflicts, between them.

3 Approximation Algorithms

3.1 An $O(n^2 + kn)$ Time 3-Approximation Algorithm for CMCT

Let $\mathcal{T}$ be a collection of trees on an n-leaf set L. It is well-known that $\mathcal{T}$ is compatible iff every pair of trees in $\mathcal{T}$ is compatible [8]. Moreover,

Lemma 3 ([4]). *$\mathcal{T}$ is a compatible collection of trees iff there exists a minimum refinement T of $\mathcal{T}$ and $rt(T) = \bigcup_{T_i \in \mathcal{T}} rt(T_i)$.*

If $\mathcal{T}$ is compatible, a minimum refinement T of $\mathcal{T}$ is a solution for MCT, as $L(T) = L$. From Lemma 2, one can obtain T by first computing a minimum refinement $T_{1,2}$ of two trees $T_1, T_2 \in \mathcal{T}$, and then iterating on $\mathcal{T} - \{T_1, T_2\} \cup \{T_{1,2}\}$ until only one tree remains that is the sought tree T.
If $\mathcal{T}$ is not compatible, then we apply the following:

Lemma 4 ([2, 3, 11]). *Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be a collection of trees on a leaf set L and let $\mathcal{C}$ be an hs-peacemaker, resp. an h-peacemaker, of $\mathcal{T}$. Then any tree in $\mathcal{T}|(L - L(\mathcal{C}))$ is a 3-approximation for* CMAST*, resp. any refinement of $\mathcal{T}|(L - L(\mathcal{C}))$ is a 3-approximation for* CMCT*, on $\mathcal{T}$.*

Given a pair of trees, [4] give an $O(n)$ time algorithm that either returns a minimum refinement when the trees are compatible, or otherwise identifies a hard conflict C between them. Thus, from Lemma 4, the procedure sketched above for a compatible collection, can be adapted to obtain a 3-approximation of CMCT for a non-compatible collection $\mathcal{T}$. Apply the algorithm of [4] to a pair of trees $\{T_1, T_2\} \subseteq \mathcal{T}$ to obtain either their minimum refinement $T_{1,2}$ or a hard conflict C. In the latter case, remove C from all input trees and iterate. In the former case, iterate on $\mathcal{T} - \{T_1, T_2\} \cup \{T_{1,2}\}$. When $\mathcal{T}$ is reduced to a single tree, $O(k + n)$ calls to the algorithm of [4] have been issued and the resulting set $\mathcal{C}$ of removed conflicts is an h-peacemaker. Hence:

Theorem 1. *The* CMCT *problem on a collection of k rooted trees on a same n-leaf set can be 3-approximated in $O(n^2 + kn)$ time.*

3.2 A Linear Time 3-Approximation Algorithm for CMAST

W.l.o.g., this section considers input trees on a same n-leaf set labelled by positive integers $1, 2, \ldots, n$. First consider collections $\mathcal{T}$ of two trees. The following characterization of isomorphic trees is the basis of our algorithm.

Lemma 5 ([4]). *Two trees T_i and T_j are isomorphic iff $rt(T_i) = rt(T_j)$ and $f(T_i) = f(T_j)$.*

The definition of $MAST(\mathcal{T})$ is independent of the order of the children of nodes in trees. However, to efficiently compute an approximation of $MAST(\mathcal{T})$, we considered that T_1 and T_2 are *ordered*. Ordering a tree T consists in totally ordering the children of every node in T. Thereby, this uniquely defines a left-right order π_T on the leaves L of T.

Given an arbitrary ordering of T_1, the approximation algorithm first tries to order T_2 accordingly. In the following, π_1, resp. π_2, stands for π_{T_1}, resp. π_{T_2}; and $\pi_2(i)$ stands for the i-th leaf in π_2. W.l.o.g., we also assume $\pi_1 = 1 \dots n$.

Definition 2. *Let π be an order on a set L. A subset S of L is an* interval *of π whenever the elements of S occur consecutively in π (but not necessarily in the same order). A tree T with leaf set L is* embeddable *in an order π on L whenever T can be ordered s.t. $\pi_T = \pi$.*

Lemma 6. *Let T be a tree with leaf set L and π be an arbitrary order of L. Then, T is* embeddable *in π iff for any node u of T, $L(u)$ is an interval of π.*

Proposition 1. *Let T be a tree and π be an order on its leaves. Testing whether T is embeddable in π costs $O(n)$ time. In the positive, ordering T such that $\pi_T = \pi$ can be done in $O(n)$ time.*

The running time stated in this proposition is achieved by performing bottom-up walks on disjoint paths in T, as described by Algorithm 1. For a node u in a tree, let $m(u)$ and $M(u)$ resp. denote the smallest and largest leaf of $L(u)$ in π. Assume the children of any non-leaf node $v \in T$ are originally stored in a doubly-linked list $l_c(v)$ which has to be ordered into a list $l'_c(v)$ so that $\pi_T|L(v) = \pi|L(v)$.

Algorithm 1: TreeOrder(T, π)

```
for any node u in T do l'_c(u) ← ∅ ;
for i = 1 to n do
    let u be the leaf s.t. u = π^{-1}(i);
    repeat
        Let v be the parent node of u in T;
        Remove u from l_c(v) and put it at the end of l'_c(v);
        u ← v;
    until i ≠ m(u) or u is the root;
```

Due to the existence of conflicting triples, two arbitrary trees T_1 and T_2 with same leaf set L may not be embeddable in a common order of L. If so, we can however show the following:

Proposition 2. *Let T_1, T_2 be trees with leaf set $L = \{1, \dots, n\}$. In time $O(n)$ it is possible to identify a set $\mathcal{C}$ of disjoint conflicts between T_1 and T_2 s.t. $T_2|\big(L - L(\mathcal{C})\big)$ is embeddable in $\pi_1|\big(L - L(\mathcal{C})\big)$.*

Below is given a sketch of the proof for this proposition. Let u be a node in a tree T with leaf set L and π be an arbitrary order on L. If an element $x \in L - L(u)$ is s.t. $m(u) <_\pi x <_\pi M(u)$, then $prev_\pi(x,u)$, resp. $next_\pi(x,u)$, stands for the maximum, resp. minimum, element of $L(u)$ w.r.t. π that is smaller, resp. larger, than x.

Lemma 7. *Let T_1, T_2 be trees on a leaf set $L \subseteq \{1,\dots,n\}$ and let $\{a,b,c\} \subseteq L$. If both $a <_{\pi_1} b <_{\pi_1} c$ and $ac \mid b \in rt(T_2)$, then $\{a,b,c\}$ is a conflict between T_1 and T_2. In particular, for a node u of T_2 and a leaf $x \notin L(u)$ s.t. $m(u) <_{\pi_1} x <_{\pi_1} M(u)$ then $\{prev_{\pi_1}(x,u), x, next_{\pi_1}(x,u)\}$ is a conflict between T_1 and T_2.*

This lemma guides the search of T_2 to remove leaves (in T_2 and T_1) forming a set of disjoint conflicts $\mathcal{C}$ s.t. for any node u of $T_2|(L - L(\mathcal{C}))$, $L(u)$ is an interval of leaves in $\pi_1|(L - L(\mathcal{C}))$. Such a node u is then said to be *full*. When all nodes of the resulting T_2 are full, Lemma 6 ensures that T_2 is embeddable in the left-right order of the tree $T_1|(L - L(\mathcal{C}))$.

Nodes of T_2 are processed in post-order, such that the children of a node u are known to be full when u is processed. For efficiency reasons, a list L_I of disjoint intervals of π_1 is also maintained sorted w.r.t. to π_1. L_I is initially composed of unit intervals $(\{1\},\dots,\{n\})$ corresponding to leaves of T_2. Then intervals of L_I are merged or removed while processing nodes of T_2 so as to maintain the following invariant:

Invariant 1. *Any interval of the list L_I contains the leaf set $L(u)$ of some node u of T_2 that is full w.r.t. $\pi_1|\big(L - L(\mathcal{C})\big)$.*

When a non-full node u is processed in the traversal of T_2, this invariant together with pointers from each children of u to the corresponding elements ordered in L_I enables us (according to Lemma 7) to efficiently identify conflicts whose removal turns u into a full node. Note that Invariant 1 is robust under the removal of a leaf in $L(v)$ for any processed node v.

Lemma 8. *Let T_1, T_2 be two trees with leaf set in L and u be the current node of T_2 to be processed by the bottom-up algorithm (*i.e. *the children of u are full w.r.t. π_1). Then a set $hs(u)$ of disjoint conflicts between $\{T_1, T_2\}$ s.t. u is full w.r.t. $\pi_1|\big(L - L(hs(u))\big)$ is found in time $O(d^+(u) + |hs(u)|)$.*

Proposition 2 follows from Lemma 7, Invariant 1 and Lemma 8. Given two arbitrary trees T_1, T_2, propositions 1 and 2 show that, in linear time, disjoint conflicts can be removed and children of nodes in T_2 ordered s.t. the two resulting trees have the same left-right order on their leaves. Thus, from now on, assume that $\pi_1 = \pi_2$. For convenience, even if some leaves have been removed, we note $\pi_1 = 1 \dots n$. Even if T_1 and T_2 have the same left-right order on their leaves, they may still host conflicting triples. However, let us show that a post-order search of T_1 (or equiv. T_2) is sufficient to remove such conflicts.

Definition 3. *Let u be a node in a tree T, then $rt(u)$ is the subset of triples $x|yz \subseteq rt(T)$ s.t. $\#\big(\{x,y,z\} \cap L(u)\big) \geq 2$, and $f(u)$ is the set of fans $\{x,y,z\} \subseteq f(T)$ s.t. $\{x,y,z\} \subseteq L(u)$. Define a node u in tree T_1 to be* valid *w.r.t. tree T_2 if both $rt(u) \subseteq rt(T_2)$ and $f(u) \subseteq f(T_2)$ hold.*

Note that if r_1 is the root node of tree T_1, then $rt(r_1) = rt(T_1)$ and $f(r_1) = f(T_1)$. Moreover, given a tree T_2 s.t. $L(T_2) = L(T_1)$, the validity of r_1 w.r.t. T_2 implies that T_1 and T_2 are isomorphic, as any 3-leaf set is either a rooted triple or a fan of both trees. Next lemma is the basis of a recursive process to obtain the validity of r_1 w.r.t. T_2.

Lemma 9. *Let u be a node of T_1 whose children, denoted $c_1, \ldots, c_{d^+(u)}$, are all valid. Let $p(m(u))$, resp. $s(M(u))$, be the leaf preceding $m(u)$, resp. succeeding $M(u)$, in π_2 if it exists.*

1. *if $\{p(m(u))|m(u)M(u), s(M(u))|m(u)M(u)\} \subseteq rt(T_2)$ then $rt(u) \subseteq rt(T_2)$*
2. *if u has only two children then $f(u) \subseteq f(T_2)$*
3. *if u has at least three children and for any $i \in \{1, 2, \ldots, d^+(u) - 2\}$, $\{m(c_i), m(c_{i+1}), m(c_{i+2})\} \in f(T_2)$, then $f(u) \subseteq f(T_2)$.*

Lemma 9 implies that if every node $u \in T_1$ is processed after its children, examining only $O(d^+(u))$ 3-leaf sets is enough to know whether a node $u \in T_1$ is already valid. When a conflict is encountered during this examination, its leaves are removed from the trees.

Indeed, thanks to Lemma 1, removing a leaf in $S(u)$ does not change the pre-established validity of inner nodes of $S(u)$. Thus, if $c(u)$ denotes the number of such encountered conflicts, ensuring the validity of u involves looking at $O(d^+(u) + c(u))$ 3-leaf sets. See Algorithm 2 for a complete description of the procedure. Note that persistent dummy leaves can be artificially added at the beginning and end of π_1 and π_2 s.t. $p(m(u))$ and $s(M(u))$ always exist for any processed node u. Processing the whole tree T_1 globally involves $O(n)$ 3-leaf sets as $\sum_{u \in T_1} c(u) = O(n)$ and $\sum_{u \in T_1} d^+(u) = O(n)$.

Provided π_1 is stored in a doubly-linked list; symmetric pointers are maintained between a node $u \in T_1$ to be processed, and the two elements of π_1 that are the leftmost and rightmost leaves of $S(u)$; and T_2 is preprocessed so as to identify in $O(1)$ the least common ancestor of any two of its nodes; then Algorithm 2 runs in linear time. Hence,

Theorem 2. *The* CMAST *problem on a collection of k rooted trees with same n-leaf set can be 3-approximated in $O(kn)$ time.*

The reader should notice that the above algorithms can be realized simultaneously by a single search of the tree. According to Proposition 1, Proposition 2 and Algorithm 2, the case $k = 2$ is solved in $O(n)$ time. Handling a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of $k > 2$ trees is done as for the MCT problem (see Section 3.1), i.e. by successively considering pairs of trees in $\mathcal{T}$. This procedure runs in $O(nk)$ and, from Lemma 4, provides a 3-approximation of CMAST for $\mathcal{T}$.

4 Inapproximability Results for MAST and MCT

In this section, we first state that the rooted and unrooted versions of CMAST (equiv. CMCT) have the same approximation threshold. Then we detail new negative results concerning the approximation of MCT, CMAST and CMCT.

Algorithm 2: AGREEMENTSUBTREE (T_1, T_2)

Input: Two rooted trees s.t. $\pi_1 = \pi_2$
for *each node* u *in a post order traversal of* T_1 **do**
  /* *Ensures that* $rt(u) \subseteq rt(T_2)$ */
  repeat
    $m(u) \leftarrow$ leftmost leaf of $S(u)$; $M(u) \leftarrow$ rightmost leaf of $S(u)$
    $p(m(u)) \leftarrow$ leaf preceding $m(u)$ in π_1 ; $f(M(u)) \leftarrow$ leaf following $M(u)$ in π_1
    if $p(m(u))|m(u)M(u) \notin rt(T_2)$ **then** remove $p(m(u))$, $m(u)$, $M(u)$ from T_1 and T_2
    else if $f(M(u))|m(u)M(u) \notin rt(T_2)$ **then** remove $f(M(u))$, $m(u)$, $M(u)$ from T_1 and T_2
  until $\{p(m(u))|m(u)M(u), f(M(u))|m(u)M(u)\} \subseteq rt(T_2)$ *or* $d^+(u) < 2$
  /* *Ensures that* $f(u) \subseteq f(T_2)$ */
  $i \leftarrow 1$
  while $d^+(u) > 2$ *and* $i \leq d^+(u) - 2$ **do**
    let $c_1, c_2, \ldots, c_{d^+(u)}$ be the children of u
    if $\{m(c_i), m(c_{i+1}), m(c_{i+2})\} \in f(T_2)$ **then** $i \leftarrow i + 1$
    else remove $m(c_i)$, $m(c_{i+1})$, $m(c_{i+2})$ from T_1 and T_2
return T_1

4.1 Rooted and Unrooted Versions of CMAST (equiv. CMCT) Share the Same Approximation Threshold

Let $\varphi(n, k)$ be a function in $\Omega(n \times k)$.

Proposition 3. *Let* $\rho \geq 1$ *be a real constant. Assume there exists a* ρ*-approximation algorithm for* CMAST*, resp.* CMCT*, on rooted trees with* $O(\varphi(n, k))$ *running time. Then, there exists a* ρ*-approximation algorithm for* CMAST*, resp.* CMCT*, on unrooted trees with* $O(n \times \varphi(n - 1, k))$ *running time.*

Proposition 3 is implicitely used in [11] and is proved in the following way. Let $\mathcal{U}$ be a collection of unrooted trees. To ρ-approximate CMAST, resp. CMCT, on instance $\mathcal{U}$, apply the hypothetical ρ-approximation algorithm to each collection obtained by rooting all trees in $\mathcal{U}$ at a same leaf. Then, return the best of the n computed solutions. Combining Theorem 2 and Proposition 3, resp. Theorem 1 and Proposition 3, we obtain that the unrooted version of CMAST, resp. CMCT, is 3-approximable in $O(kn^2)$, resp. $O(n^3 + kn^2)$, time. Using a simple padding argument yields the converse of Proposition 3:

Proposition 4. *Let* $\rho \geq 1$ *be a rational constant. Assume there exists a* ρ*-approximation algorithm for* CMAST*, resp.* CMCT*, on unrooted trees with* $O(\varphi(n, k))$ *running time. Then, there exists a* ρ*-approximation algorithm for* CMAST*, resp.* CMCT*, on rooted trees with* $O(\varphi(n + \lceil \rho n \rceil, k))$ *running time.*

4.2 Hardness of Approximating CMAST on Three Trees

Theorem 3. *The* 3-CMAST *problem is* APX*-hard.*

Since 2-MAST (and thus, 2-CMAST) can be exactly solved in polynomial time [21], Theorem 3 is somehow tight. Its proof relies on a careful reading of [17] which states that the *general* 3-MAST problem is APX-hard. In fact [17] proves that a *restriction* of 3-MAST to a certain set of instances is APX-hard. CMAST is not considered in [17], but it is easy to see that for this particular set of instances, 3-MAST L-reduces to 3-CMAST

4.3 Hardness of Approximating MCT and CMCT on Two Trees

In order to prove Theorems 5 (APX-hardness of 2-CMCT) and 6 (inapproximability of 2-CMCT), we define an intermediate problem, called Maximum Star-Forest (MSF). Let $G = (V, E)$ be a graph. A *star-forest* of G is a subset of E which does not contain any path of length 3. The MSF problem is: *"given a graph G, find a star-forest of G that is of maximum cardinality"* For each integer $\Delta \geq 1$, we denote by Δ-MSFB the restriction of MSF to *bipartite* input graphs having *maximum degree* at most Δ. The restriction of the Maximum Independent Set (shortly MIS) to input graphs having maximum degree at most 3 is denoted 3-MIS. Note that 3-MIS is APX-complete [1].

Theorem 4. *The* 4-MSFB *problem is* APX*-hard.*

Proof (sketch). We use an L-reduction from 3-MIS to 4-MSFB relying on the following transformation. Let $G = (V, E)$ be an instance of 3-MIS (*i.e.* a graph with maximum degree at most 3), we construct an instance $G' = (V', E')$ of 4-MSFB as follows.

$$V' := V \cup \{\gamma_e : e \in E\} \cup \{\sigma_v, \tau_v : v \in V\},$$
$$E' := \big\{\{u, \gamma_e\}, \{\gamma_e, v\} : e = \{u, v\} \in E\big\} \cup \big\{\{v, \sigma_v\}, \{\sigma_v, \tau_v\}, : v \in V\big\}.$$

Clearly, G' can be obtained from G in polynomial time, and $\#V' = m + 3n$ and $\#E' = 2m + 2n$, where n and m denote the cardinality of V and E resp.

□

Theorem 4 leads to the following result:

Proposition 5. 2-MCT *is* APX*-hard even if it is restricted to collections $\mathcal{T}$ of two rooted trees satisfying $\#MCT(\mathcal{T}) \geq \frac{1}{4} \times n$, where n denotes the size of each tree in $\mathcal{T}$.*

Proof (sketch). We use an L-reduction from 4-MSFB to 2-MCT relying on the following transformation. Let $G = (V, E)$ be an instance of 4-MSFB. Since G is bipartite there exists two independent sets I_1 and I_2 of G partitioning V. W.l.o.g., we can assume that G has no isolated vertex. We construct a collection $\mathcal{T} = \{T_1, T_2\}$ of two rooted trees with leaf set E. The root of T_i is denoted r_i. For each $v \in I_i$, let X_v be the non-empty *star-tree* whose leaf set is the set of all edges of E admitting v as an extremity (a *star-tree*, is a fan with an arbitrary number of leaves). The child subtrees of r_i, are trees X_v with $v \in I_i$.

The transformation requires polynomial time and the size of the instance of 2-MCT in linear in the size of the instance of 4-MSFB. The correctness of the reduction follows by proving that for each subset $F \subseteq E$, F is a star-forest of G iff $T_1|\,F$ and $T_2|\,F$ are compatible. □

Proposition 5 yields the two main results of this section. On the first hand, we obtain:

Theorem 5. *The* 2-CMCT *problem is* APX*-hard.*

On the other hand, using the "self-improvement" technique of [17, 19] we deduce from Proposition 5 that 2-MCT is hard to approximate within constant ratio:

Theorem 6. *For any real constant* $\delta < 1$*, the* 2-MCT *problem cannot be approximated within ratio* $2^{\log^{\delta} n}$*, unless* $\mathrm{NP} \subseteq \mathrm{DTIME}\big(2^{\mathrm{polylog}\, n}\big)$.

The analoguous to Theorem 6 for 3-MAST is [17, Theorem 3].

4.4 Hardness of Approximating MCT on Unbounded Number of Trees

For the general MCT problem we can find non-approximability results stronger than Theorem 6. Approximating MCT on collections of n-leaf trees is at least as hard as approximating MIS on n-vertex graphs. The proof consists in an approximation preserving reduction from MIS to MCT, similar to the reduction from MIS to MAST described in [5]. Since MIS is very hard to approximate [16] (see also [9]), we obtain:

Theorem 7. *For all real* $\epsilon > 0$*,* MCT *is not approximable within ratio* $(1+\epsilon)n^{1-\epsilon}$ *unless* $\mathrm{NP} = \mathrm{ZPP}$*, resp. within ratio* $(1+\epsilon)n^{0.5-\epsilon}$ *unless* $\mathrm{P} = \mathrm{NP}$.

Note that Theorem 7 still holds if MCT is restricted to collections of trees containing at least a binary tree. Remark that using the *approximating via partitioning* paradigm [14], one can approximate MAST within $n/\log n$ [18]. This also holds for MCT.

References

1. P. Alimonti and V. Kann. Some APX-completeness results for cubic graphs. *Theor. Comput. Sci.*, 237(1–2):123–134, 2000.
2. A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithm. *SIAM J. on Comput.*, 26(6):1656–1669, 1997.
3. V. Berry and F. Nicolas. Maximum agreement and compatible supertrees. In *15th Annual Symposium on Combinatorial Pattern Matching (CPM'04)*, volume 3109 of *LNCS*, pages 205–219, 2004.
4. V. Berry and F. Nicolas. Improved parametrized complexity of maximum agreement subtree and maximum compatible tree problems. *IEEE Trans. on Comput. Biology and Bioinf.*, (to appear).

5. P. Bonizzoni, G. Della Vedova, and G. Mauri. Approximating the maximum isomorphic agreement subtree is hard. *Int. J. of Found. of Comput. Sci.*, 11(4):579–590, 2000.
6. D. Bryant. *Building trees, hunting for trees and comparing trees: theory and method in phylogenetic analysis.* PhD thesis, University of Canterbury, Department of Mathemathics, 1997.
7. R. Cole, M. Farach-Colton, R. Hariharan, T. M. Przytycka, and M. Thorup. An $O(n \log n)$ algorithm for the Maximum Agreement SubTree problem for binary trees. *SIAM J. on Comput.*, 30(5):1385–1404, 2001.
8. G. F. Eastabrook and F. R. McMorris. When is one estimate of evolutionary relationships a refinement of another? *J. of Math. Biol.*, 10:367–373, 1980.
9. L. Engebretsen and J. Holmerin. Towards optimal lower bounds for clique and chromatic number. *Theor. Comput. Sci.*, 299(1–3):537–584, 2003.
10. M. Farach, T. M. Przytycka, and M. Thorup. On the agreement of many trees. *Inf. Proces. Letters*, 55(6):297–301, 1995.
11. G. Ganapathy and T. J. Warnow. Approximating the complement of the maximum compatible subset of leaves of k trees. In *5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'02)*, volume 2462 of *LNCS*, pages 122–134, 2002.
12. G. Ganapathysaravanabavan and T. J. Warnow. Finding a maximum compatible tree for a bounded number of trees with bounded degree is solvable in polynomial time. In *1st Int. Workshop on Algorithms in Bioinformatics (WABI'01)*, volume 2149 of *LNCS*, pages 156–163, 2001.
13. A. Gupta and N. Nishimura. Finding largest subtrees and smallest supertrees. *Algorithmica*, 21(2):183–210, 1998.
14. M. M. Halldòrsson. Approximations of weighted independent set and hereditary subset problems. *J. of Graph Algor. and Appl.*, 4(1), 2000.
15. A. M. Hamel and M. A. Steel. Finding a maximum compatible tree is NP-hard for sequences and trees. *Appl. Math. Letters*, 9(2):55–59, 1996.
16. J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.*, 182:105–142, 1999.
17. J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Disc. Appl. Math.*, 71(1–3):153–169, 1996.
18. J. Jansson, J. H.-K. Ng, K. Sadakane, and W.-K. Sung. Rooted maximum agreement supertrees. In *6th Latin American Symposium on Theoretical Informatics (LATIN'04)*, volume 2976 of *LNCS*, pages 499–508, 2004.
19. T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM J. on Comput.*, 24(5):1122–1139, 1995.
20. M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting. A decomposition theorem for maximum weight bipartite matchings with applications to evolutionary trees. In *7th Annual European Symposium on Algorithms (ESA'99)*, volume 1643 of *LNCS*, pages 438–449, 1999.
21. M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *J. of Algor.*, 40(2):212–233, 2001.
22. M. A. Steel and T. J. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Inf. Proces. Letters*, 48(2):77–82, 1993.

Theoretically Good Distributed CDMA/OVSF Code Assignment for Wireless Ad Hoc Networks

Xiang-Yang Li[1,⋆] and Peng-Jun Wan[1,2,⋆]

[1] Illinois Institute of Technology, Chicago, IL, USA
{xli,wan}@cs.iit.edu
[2] Hong Kong City University, Hong Kong, China

Abstract. We present several distributed CDMA/OVSF code assignment algorithms for wireless ad hoc networks modelled by unit disk graph (UDG). We first give a distributed code assignment whose total throughput is within a constant factor of the optimum. Then we give a distributed method such that the minimum rate achieved is within a constant factor of the optimum. A distributed method that can approximate both the minimum rate and total throughput is also presented. All our methods use only $O(n)$ total messages (each with $O(\log n)$ bits) for an ad hoc wireless network of n nodes modelled by UDG.

Keywords: CDMA code assignment, coloring, throughput, bottleneck, interference, wireless networks.

1 Introduction

We consider a static wireless ad hoc network consisting of a set V of n nodes distributed in a two-dimensional plane. Assume all nodes have the same transmission radius r, thus, wireless ad hoc networks are modelled by unit disk graphs (UDG), in which two nodes are connected iff their Euclidean distance is no more than r. We assume that the omnidirectional antenna is used by all wireless nodes: the signal sent by a node will be received by all nodes inside its transmission region. The transmission region of a node u is thus modelled as a disk $D(u, r)$ centered at u with radius r. To increase the capacity of the network, frequency spectrum has to be reused as it is one of the scarcest resources available. Same channel is not assigned to two nodes if it causes either *primary interference* or *secondary interference*. Primary interference occurs if two nodes use the same channel and one is inside the transmission region of the other. The secondary interference occurs if a third node is within the common transmission regions of two nodes using the same frequency channel. The *interference graph* $G = (V, E)$ has an edge uv if two nodes u and v will generate interference when they are assigned the same channel. Assigning frequency channel efficiently in UDG has been well-studied [6, 10] but little is known about assigning CDMA/OVSF code for wireless ad hoc networks while achieving some global quality such as maximizing the total throughput or the bottleneck of the networks.

⋆ The research was supported in part by NSF under Grant CCR-0311174.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 126–135, 2005.

In a CDMA system, the channels are defined by the pseudo-random codewords. For simplification, we represent each CDMA/OVSF codeword by a binary string (called colors hereafter). Two colors are said to be *prefix-free* if neither is a prefix of the other, which is equivalent to that the corresponding codewords are orthogonal. The rate of an ℓ-bit color is equal to $2^{-\ell+1}$, which is equal to the rate of the corresponding codeword. We also say that an ℓ-bit color is in the ℓ-th layer of the CDMA/OVSF code tree structure. The root has layer 1.

A (proper) vertex coloring is to assign each vertex a color such that two adjacent vertices receive different colors. The CDMA code assignment is to assign colors to nodes such that adjacent nodes in the interference graph receive prefix-free colors, which is called *prefix-free* vertex coloring. The minimum vertex coloring of the interference graph has been studied in the context of channel assignment in wireless ad hoc networks channelized by FDMA, TDMA or CDMA/OVSF [2–5, 7, 11, 12, 14, 15]. The majority of these CDMA code assignment methods simply presented networking protocols to obtain a proper vertex coloring without addressing the computational complexity and/or the optimization. Sen and Huson [13] gave a proof of the NP-hardness of the vertex coloring in interference graph even when all nodes are located in a plane and have the same transmission radii. A problem related to the (prefix-free) vertex coloring of the interference graphs is the *distance-2 vertex coloring* [8]. A *distance-2 vertex coloring* of a graph H is a proper vertex coloring of H^2, the *square graph* of H, which is the graph obtained by creating an edge between each pair of vertices of H separated by at most two hops in H. Notice that the colors assigned to two adjacent nodes in H^2 should only be different for a vertex coloring problem, while these two colors should further be prefix-free for CDMA/OVSF code assignment.

Given a prefix-free CDMA code assignment $\{c_v \mid v \in V\}$, its *throughput* and *bottleneck* are defined as $\sum_{v\in V} 2^{-|c_v|+1}$ and $\min_{v\in V} 2^{-|c_v|+1}$ respectively, where $|c_v|$ denotes the number of bits of the color c_v. The *throughput of an interference graph* G, denoted by $\tau(G)$, is then the maximum of the throughput over all prefix-free code assignments of G. Similarly, the *bottleneck of an interference graph* G, denoted by $\beta(G)$, is then the maximum of the bottleneck over all prefix-free code assignments of G. We will study various optimization problems on prefix-free vertex coloring of the interference graphs: maximize the total throughput, the minimum rate, and both at the same time.

The main contributions of this paper are as follows. We propose several efficient distributed CDMA/OVSF code assignment algorithms for wireless ad hoc networks modelled by UDG. We first study how to assign CDMA/OVSF code such that the total throughput achieved is within a constant factor of the optimum. Then we give a method such that the minimum rate achieved is within a constant factor of the optimum. A method that can approximate both the minimum rate and total throughput simultaneously is also presented. All our methods use only $O(n)$ total messages (each with $O(\log n)$ bits) for an ad hoc wireless network of n nodes modelled by UDG.

2 Distributed Code Assignment

Let $N_k(u)$ be the set of all wireless nodes that are at most k hops away from node u in UDG, $d_k(u)$ be the cardinality of $N_k(u)$. Obviously, nodes that can have primary interference with u are $N_1(u)$ only; nodes that can have either primary interference or secondary interference with u are $N_2(u)$ only. If every node knows its exact geometry location, a communication efficient protocol [1] is known to find all two-hop neighbors of all nodes using at most $O(n)$ communications.

2.1 Maximize Throughput $\tau(G)$

First-fit coloring is a class of greedy algorithms for vertex coloring. Assume that there is a (partial) ordering of all nodes. We then assign code to nodes sequentially according to the associated ordering by assigning each node the shortest possible code. Thus, in any first-fit coloring, all nodes receiving the same code form a *maximal* independent set (MIS). Intuitively, such MIS should be a small constant approximation of a *maximum* independent set to maximize the throughput. Clearly, the performance of a first-fit code assignment depends on the ordering used. Indeed, there always exists an ordering in which the first-fit coloring generates an optimal code assignment. However, such ordering is unlikely to be found in polynomial time due to the expected NP-hardness of the max-throughput code assignment. So we seek some node ordering that produces a code assignment approximating $\tau(G)$; such node ordering should be generated efficiently. We propose several different node orderings and show that all of them produce a code assignment with total throughput $O(\tau(G))$ and use total communications $O(n)$. Hereafter, we assume that each message has $O(\log n)$ bits. All node orderings used here are just partial ordering computed locally.

We will first construct the interference graph and then construct an MIS based on a rank (*e.g.*, ID, or degree, or the node's geometry position). Nodes in the MIS are assigned the shortest code 10. For the remaining nodes, we assign code using the first fit heuristics based on a partial ordering (*e.g.*, ID, or degree, or the node's geometry position). Here we assume that every node has a distinctive ID and knows its geometry position if a communication efficient protocol is needed. Algorithm 1 presents our method (run by every node u) of assigning CDMA/OVSF code based on ordering by ID to maximize the throughput.

We find an MIS in a distributed manner as follows: initially all nodes' status are *White*; a node becomes *InIS* if it has a *rank* smaller than all its neighbors with status *White*; a node becomes *NotInIS* if it has an *InIS* neighbor. A node could be either *White*, or *InIS*, or *NotInIS*. The nodes with status *InIS* form an MIS. We will show that its size is within a constant factor of the maximum independent set for an interference graph G. See [9, 17] for more details.

Obviously Algorithm 1 generates a prefix-free code assignment since, for each pair of neighboring nodes u and v in the interference graph, the node with larger ID can only assign code after it gets the code of the other node. The total communication cost is $O(n)$ since we use communication efficient protocol to

Algorithm 1 Max-Throughput Using ID-ordering by u

1: Node u sends a message to tell its ID to all nodes $N_1(u)$. If secondary interference is not permitted, node u finds $N_2(u)$ using a communication efficient method [1].
2: All nodes collectively find an MIS based on a rank ID.
3: Node u assigns a CDMA code represented by binary 10 if u is in the MIS. It then informs its neighbors in G about its code.
4: If node u receives a CDMA code from its neighbor in G, u marks the corresponding code *used* in the CDMA/OVSF tree structure stored locally.
5: Then assign code to the remaining nodes. If node u has an ID smaller than all its neighbors in G without a code, then node u finds the smallest layer $h > 0$ in the local CDMA/OVSF code tree such that layer h has at least 2 free codes not marked used. Node u picks the first unused code in layer h and informs its neighbors in G about its code. The picked code is called the *first fit* code for node u.

collect $N_2(u)$ for all nodes and to inform the assigned code to its neighbors in G. We can also use the node degree (or position) to find an MIS in Algorithm 1.

We then show that the above methods indeed approximate the optimum throughput $\tau(G)$. To do so, we first study the structure of some optimum CDMA code assignment, called *canonical coloring*. In [16], we defined the *canonical coloring* as follows. Given a graph $G = (V, E)$, partition the vertex set V into independent sets V_1, V_2, $\cdots$, V_k with $|V_1| \geq |V_2| \geq \cdots \geq |V_k|$. Let $G_0 = G$ and G_i be the graph of removing the vertices V_i and the incident edges from graph G_{i-1}, for $1 \leq i \leq k$. Vertex set V_i is a maximum independent set of graph G_{i-1}. For $1 \leq i \leq k-1$, all nodes in V_i receive the code 1^i0, and all nodes in V_k receive the code 1^k. Obviously, the throughput of such canonical coloring is $\sum_{i=1}^{k-1} \frac{|V_i|}{2^i} + \frac{|V_k|}{2^{k-1}}$. Notice that, If there are multiple maximum independent sets V_1, we have to choose the one that produces the largest maximum independent set V_2. Similarly, the selection of the first i maximum independent sets V_1, V_2, $\cdots$, V_i produces the largest maximum independent set V_{i+1}, for $1 \leq i < k$. Call such sequence of maximum independent set as *canonical maximum independent set decomposition* and the corresponding coloring *canonical coloring*.

Theorem 1. *[16] The canonical coloring maximizes the throughput.*

This theorem implies that the maximum throughput of any code assignment is at most the independence number $\alpha(G)$ of the interference graph G. Based on this observation, we can assign the code as follows. First, compute an MIS that approximates the maximum independent set (with approximation ratio ϱ). Then assign the nodes in the MIS a code 10 (its rate is 1/2). For the remaining nodes, we can recursively find the MIS and assign code 1^i0 for the MIS retrieved in the ith iteration but the messages of this approach could be very large. To optimize the message complexity, Algorithms 1 used a different approach for the remaining nodes (actually any prefix-free code assignment for the remaining nodes works here). Obviously, the throughput generated by assigning nodes in MIS a code 10 is at least $\varrho \cdot \alpha(G)/2$. This implies the following theorem (see appendix for the proof).

Theorem 2. *Algorithm 1 generates a code assignment whose throughput is at least $\varrho/2$, where $\varrho = 1/5$ if only primary interference is concerned and $\varrho = 1/13$ if secondary interference is also concerned.*

When every node knows its position, we can further improve the theoretical lower bounds on the throughput of the assigned codes as follows. We still construct an MIS first, but instead of using the node ID or the degree as selection criterion, we select a node u to the MIS if all unassigned neighboring nodes are inside one half of the disk centered at u. Notice that such node u always exists since the most left undecided node trivially satisfies this condition.

Algorithm 2 Max-Throughput Using Position-ordering by u

1: Every node finds its neighbors in G using a communication efficient protocol in [1].
2: All nodes together compute an MIS based on rank $(x(u), y(u), ID(u))$, where $x(u)$, and $y(u)$ are the x-coordinate and y-coordinate of u.
3: Node u gets code 10 if it is in the computed MIS.
4: All nodes not in MIS get the *first fit* code in an *increasing* ordering of $(x(u), y(u), ID(u))$ using method similar to the last step of Algorithm 1.

Theorem 3. *Algorithm 2 generates a code assignment whose throughput is at least $\varrho/2$, where $\varrho = 1/3$ ($\varrho = 1/7$ resp.) if primary interference (secondary interference resp.) is concerned. It uses $O(n)$ messages, each with $O(\log n)$ bits.*

The proof is similar to Theorem 2 and is omitted. The approximation ratio could be further improved to be better than $\varrho/2$, which is analyzed as follows. The new approach will compute an MIS V_1', and then compute an MIS V_2' for the remaining nodes. Clearly, the number of messages is still $O(n)$. The nodes in V_1' will receive a code 10 and the nodes in V_2' will receive a code 110. We assign codes to other nodes using a method similar to the last step of Algorithm 1.

Theorem 4. *An ϱ-approximation algorithm for the maximum independent set gives a $\frac{5}{8}\varrho$-approximation algorithm for the maximum throughput code assignment.*

Proof. Consider a canonical maximum independent decomposition $V_1, V_2, \cdots, V_k$ of all nodes V. Here $|V_1'| \geq \varrho \cdot |V_1|$. Let $t_{i,j} = \frac{|V_i' \cap V_j|}{|V_j|}$, i.e., the portion of V_j is used in V_i'. After V_1' is generated, we know that the maximum independent set in the remaining graph has size at least $\max((1-t_{1,1}) \cdot |V_1|, (1-t_{1,2}) \cdot |V_2|)$, since $V_1 - V_1' \cap V_1$ and $V_2 - V_1' \cap V_2$ are still independent sets. Notice that $t_{1,1} \cdot |V_1| + t_{1,2} \cdot |V_2| \leq V_1$. Then $(1-t_{1,1}) \cdot |V_1| + (1-t_{1,2}) \cdot |V_2| \geq |V_2|$. It implies that V_2' has size at least $\varrho \cdot |V_2|/2$. Consequently, the throughput τ' generated by partition $V_1', V_2', \cdots, V_k', \cdots, V_{k_2}'$ is at least $\varrho \cdot (\frac{|V_1|}{2} + \frac{|V_2|}{2 \cdot 2^2})$. Remember that the canonical coloring has throughput τ at most $\frac{|V_1|}{2} + 2 \cdot \frac{|V_2|}{2^2}$ using fact $|V_i| \leq |V_2|$. From $|V_2| \leq |V_1|$, it is easy to show that $\tau' \geq \frac{5}{8}\varrho \cdot \tau$. This finishes the proof. □

Theorem 5. *If node position is known, we can produce a code assignment, using $O(n)$ total messages, whose total throughput is at least $5/24$ (resp. $5/56$) of the optimum when primary interference (reps. secondary interference) is concerned.*

2.2 Maximize Bottleneck $\beta(G)$

We continue to study how to assign codes to maximize the minimum rate. Intuitively, to maximize the throughput, from the canonical code assignment discussion, the assigned codes should be imbalanced. However, to maximize the minimum rate, the assigned codes should be as balanced as possible. Clearly, the previous greedy methods do not generate a balanced code assignment. In this section, we present a novel distributed method to assign a balanced code.

Our method is based on the following observation. Consider a node u and all its neighbors in the interference graph G. If all such neighbors and u form a clique, then the minimum rate of these nodes is approximately $1/d$, where d is the size of the clique. This is achieved when all nodes use the code in level $\log d$. In other words, to maximize the minimum rate assigned, node u cannot choose the first fit code; it has to use a code in level close to $\log d$. Putting in other way, node u cannot be too greedy and it has to leave good codes for its neighbors. The following Algorithm 3 details our method.

Algorithm 3 Max-Bottleneck by Degree-ordering by a node u

1: All nodes together compute the interference graph G. Assume that each node u knows its degree $d(u)$ in G. Each node u informs its neighbors in G its degree $d(u)$.
2: Node u constructs a local binary code tree T.
3: If node u has the *largest* degree $d(u)$ among all neighbors in G without a code, where ties are broken by smaller ID, node u picks the first unmarked code in the code tree T stored locally from layer ℓ, where $2^{\ell-2} < d(u)+1 \leq 2^{\ell-1}$. Here a code is marked if it is either marked as *used* or *conflicted*.
Node u informs its neighbors in G the selected code of u efficiently.
4: If a node u receives a code message from its neighbor in G, u marks the corresponding code *used* in T, and marks all prefix-codes of this code *conflicted* in T.

Theorem 6. *Algorithm 3 generates a prefix-free code assignment whose minimum rate is within a constant factor of optimum.*

Proof. It is easy to show that it generates a prefix-free code assignment (thus the proof is omitted due to space limit). Consider a node u with the largest degree $d(u)$ in G. If primary interference is concerned, we partition the disk $D(u,1)$ into 6 equal-sized sectors. If secondary interference is also concerned, we partition the disk $D(u,2)$ into 13 equal-sized sectors. We already showed that all neighbors of u inside one sector form a complete subgraph in G. Using the pigeonhole principle, it is easy to show that among the neighbors of u in G and u, the minimum clique size is at least $c \cdot d(u) + 1$, where $c = 1/6$ for primary interference graph, and $c = 1/13$ for secondary interference graph. For a clique

of size q, the minimum rate of nodes in the clique is obviously at most $2^{-\lceil \log_2 q \rceil}$. Thus, for any assignment, the minimum rate among neighbors of u and node u is at most $2^{-\lceil \log_2 (c \cdot d(u)+1) \rceil}$. Obviously, the rate by our approach is $2^{-\lceil \log_2 (d(u)+1) \rceil}$. Then $2^{-\lceil \log_2 (d(u)+1) \rceil} \geq 2^{-\lceil \log_2 c \rceil} \cdot 2^{-\lceil \log_2 (c \cdot d(u)+1) \rceil}$ implies that the minimum rate achieved by Algorithm 3 is at least 1/8 (1/16 resp.) of the optimum if the primary interference (the secondary interference resp.) is concerned. □

2.3 Maximize $\tau(G)$ and $\beta(G)$

As we discussed before, to maximize the throughput, the assigned codes should be as imbalanced as possible, while to maximize the bottleneck rate, the assigned codes should be as balanced as possible. It seems impossible to have a code assignment that approximates both the total throughput and the bottleneck rate. In this subsection, we show that by retreating little bit on both requirements, we can achieve this. Our method is almost a straightforward combination of previous methods. We first assign the shortest code to the nodes in an MIS. For the remaining nodes, we assign a balanced code.

Algorithm 4 Max-Throughput and Bottleneck by a node u

1: All nodes together compute the interference graph G. Each node u computes its degree $d(u)$ in G and informs its neighbors in G about its degree $d(u)$.
2: All nodes together compute an MIS based on the rank by degree. Node ID or $(x(u), y(u), ID(u))$ can also be used as the rank criterion. Node u gets CDMA/OVSF code 10 if it is in the computed MIS. The remaining steps will assign code for other nodes.
3: Each node u constructs a binary code tree T.
4: If node u is not assigned and has the largest degree $d(u)$ among all its neighbors in G without a CDMA code, node u picks the first unmarked code from layer ℓ in T, where $2^{\ell-3} < d(u) \leq 2^{\ell-2}$. Node u informs all its neighbors in G the selected code.
5: If node u receives a message from its neighbor v informing the code of v, u marks this code *used*, and marks all prefix-codes of this code *conflicted* in tree T.

Theorem 7. *Algorithm 4 generates a prefix-free code assignment whose total throughput is within $\varrho/2$ of the optimum, and whose minimum rate is within $2^{-\lceil \log_2 c \rceil - 1}$ factor of the optimum, where ϱ is the approximation ratio of the maximum independent set algorithm, $c = 1/6$ for primary interference and $c = 1/13$ for secondary interference.*

3 Conclusion

We presented several efficient distributed CDMA/OVSF code assignment algorithms. Notice that our theoretical analysis is pessimistic. In [18], we presented methods to further improve the performance. We conducted extensive simulations and we found that the practical performances of our methods are much

better than these pessimistic analysis. Our methods can also be used to generate prefix-free code assignment for wireless ad hoc networks that are not modelled by UDGs. The UDG network model only enables us to prove that our methods have constant approximation ratios. However, it is unclear how to bound the communications in no-UDG model or without position information.

This paper is not intended to solve all critical issues in CDMA based wireless ad hoc networks. There are several other important issues that should be addressed, *e.g.*, the mobility of wireless nodes; and the time synchronization among the mobile wireless nodes. See [18] for more discussions.

References

1. Călinescu, G. Computing 2-hop neighborhoods in ad hoc wireless networks, 2003. AdHocNow.
2. Chlamtac, I., and Farago, A. Making transmission schedules immune to topology changes in multi-hop packet radio networks. *IEEE/ACM Transactions on Networking 2*, 1 (1994), 23–29.
3. Chlamtac, I., and Kutten, S. A spatial reuse TDMA/FDMA for mobile multihop radio nertworks. In *IEEE INFOCOM* (1985), pp. 389–394.
4. Garcia-Luna-Aceves, J. and Raju, J. Distributed assignment of codes for multihop packet-radio networks. *IEEE MILCOM*, volume 1, pages 450-454, 1997.
5. Goldberg, A., and Rao, S. Flows in undirected unit capacity networks. Tech. Rep. 97-103, NEC Research Institute, Inc, 1997.
6. Graf, A., Stumpf, M., and Weisenfels, G. On coloring unit disk graphs. *Algorithmica 20*, 3 (1998), 277–293.
7. Hu, L. Distributed code assignments for CDMA packet radio networks. *IEEE/ACM Transactions on Networking*,1:668, Dec. 1993.
8. Krumke, S., Marathe, M., and Ravi, S. Models and approximation algorithms for channel assignment in radio networks. *Wireless Networks 7* (2001), 567–574.
9. Li, X.-Y., and Wang, Y. Simple heuristics and PTASs for intersection graphs in wireless ad hoc networks. In *ACM DialM* (workshop of ACM MobiCom) (2002).
10. McDiarmid, C., and Reed, B. Colouring proximity graphs in the plane. *Discrete Mathematics 199* (1999), 123–137.
11. Nelson, R., and Kleinrock, L. Spatial-TDMA: A collision-free multihop channel access protocol. *IEEE Transactions on Communications 33*, 9 (1985), 934–944.
12. Ramanathan, S., and Lloyd, E. Scheduling algorithms for multi-hop radio networks. *IEEE/ACM Transactions on Networking 1* (April 1993), 166–172.
13. Sen, A., and Huson, M. L. A new model for scheduling packet radio networks. *ACM/Baltzer Journal Wireless Networks 3* (1997), 71–82.
14. Sen, A., and Malesinska, E. Approximation algorithms for radio network scheduling. In *Allerton Conf. on Comm., Contr. and Comp.* (1997), pp. 573–582.
15. Stevens, D., and Ammar, M. Evaluation of slot allocation strategies for TDMA protocols in packet radio networks. In *IEEE MILCOM* (1990), pp. 835–839.
16. Wan, P.-J., Li, X.-Y. and Frider, O. OVSF-CDMA code assignment for wireless ad hoc networks. ACM DialM (workshop of ACM MobiCom), 2004.
17. Wang, Y., and Li, X.-Y. Geometric spanners for wireless ad hoc networks. In *Proc. of 22nd IEEE ICDCS* (2002).
18. Li, X.-Y., Wan, P.-J. and Song, W.-Z. Theoretically Good Distributed CDMA/OVSF Code Assignment for Wireless Ad Hoc Networks. Tech Report, IIT, 2004, `http://www.cs.iit.edu/~xli/publications-select.htm`

Appendix (Proof of Theorem 2)

Proof. Let's consider all nodes, denoted by V_1, that receive code 10. Clearly, V_1 is independent. We will show that $|V_1|$ is within ϱ factor of $\alpha(G)$.

If only primary interference is concerned, G is the original UDG and it is well-known that the greedy method generates an MIS whose size is at least 1/5 of the maximum independent set. Obviously, the total throughput generated by our approach is at least $|V_1|/2$ and the optimum throughput is at most $\alpha(G) \leq 5|V_1|$.

If the secondary interference is concerned, we will prove that $|V_1|$ has size at least 1/13 of $\alpha(G)$ by showing that, $\forall u \in V_1$, there are at most 13 independent nodes in G. Let $D(x, r)$ be the disk centered at a point x with radius r hereafter. Consider a disk $D(u, 2)$ centered at node u with radius 2. Then all its neighbors $N_2(u)$ are inside the disk $D(u, 2)$. Partition this disk into 13 equal-sized sectors, each with angle $2\pi/13$. It is easy to show that the chord ab defined by the sector $\measuredangle aub$ has length $4\sin(\pi/13) < 1$. We will show that all neighboring nodes in one sector are connected. Consider any two nodes x and y from $N_2(u)$. We actually will prove a stronger result: any two neighbors of u in the sector $\measuredangle aub$ with $\|ab\| = 1$ are connected in the interference graph.

If x and y are inside $D(u, 1)$, then obviously $\|xy\| < 1$. Thus, x and y are connected in G. If y is inside $D(u, 1)$ but x is not, then there exists a node w connected to both x and u. Clearly, y and w are all inside $D(u, 1)$ now, thus, edge yw exists in the original unit disk graph. Thus, node w is inside the common transmission range of nodes y and x. It implies that x, y are connected in G (concerning the secondary interference).

Finally, we consider the case when both x and $y \notin D(u, 1)$. Assume that node w is connected to both x and u, and node v is connected to both y and u. See Figure 1 (a) and (d) for an illustration. We will then show that either $\|yx\| \leq 1$, or $\|yw\| \leq 1$, or $\|vx\| \leq 1$. Notice that, if any one is true, then x, y are connected in G. For the sake of contradiction, assume that $\|yx\| > 1$, $\|yw\| > 1$, and $\|vx\| > 1$. We partition the region $\measuredangle aub - \measuredangle cud$ into 6 regions. Figure 1 (b) illustrates such six partitions. Here segments ca, db, ab, eb, am have length 1.

We then prove that any two nodes in region $efa \cup mfb$ have distance at most 1 and any two nodes in region $efbha$ have distance at most 1. Consider any two nodes x and y in the region $efa \cup mfb$. If both are in the same triangle, then clearly $\|xy\| < 1$ since the triangles have side-length less than 1. Otherwise, let x' and y' be the intersection point of line xy with segment ea and segment mb respectively. Figure 1 (b) and (e) illustrate the proof that follows. Obviously, $\|xy\| \leq \|x'y'\| \leq \min(\|ey'\|, \|ay'\|)$. Note that $\|ey'\| \leq \min(\|em\|, \|eb\|) < 1$ and similarly $\|ay'\| \leq \min(\|am\|, \|ab\|) = 1$. Thus, $\|xy\| \leq 1$. Similar proof reveals that any two nodes in region $efbha$ have distance at most 1.

If node x is in region 2, then node y cannot be in region 3, 5, and 6 since we can show that otherwise $\|xy\| \leq 1$. In other words, node y must be in region 1 or 4 in this case. Similarly, if node x is in region 3, 5, or 6, node y must be in region 1 or 4 in this case. Thus, we assume that either node x or y (say x w.l.o.g) is in region 1 by symmetry. Obviously, node y cannot be inside the disk $D(x, 1)$ since we assume that $xy \notin G$. Thus, we have to place node v inside the sector

$\measuredangle cud$ but not inside the disk $D(x,1)$ and place y inside region $efmbha$ but not inside the disk $D(x,1)$ while still maintain $\|yv\| \leq 1$. We then show that this is impossible. Figure 1 (c) and (f) illustrate the proof that follows.

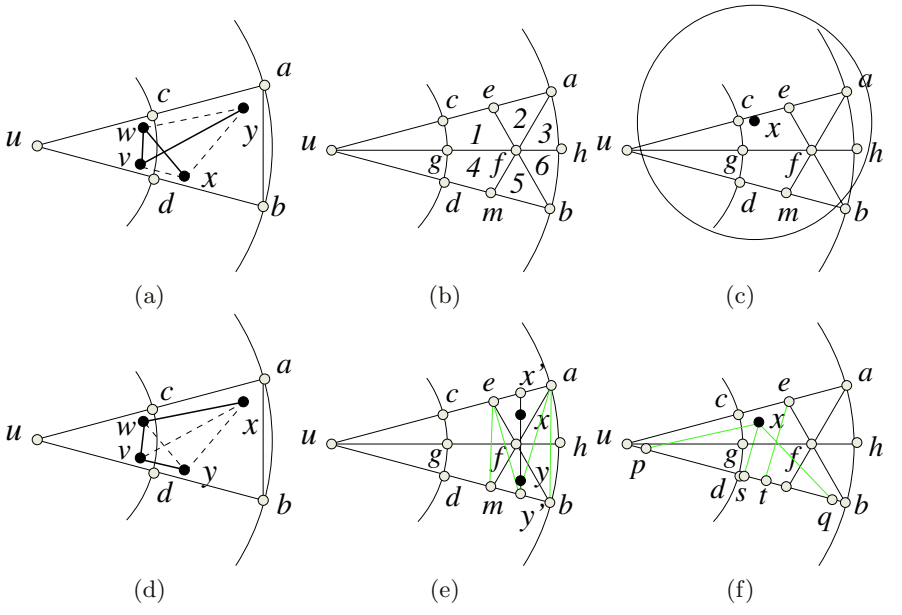


Fig. 1. All neighbors in the sector conflict with each other. Here $\|uc\| = \|ud\| = \|ab\| = 1$ and $\|ua\| = \|ub\| = 2$. (a): wx and vy intersect; (b) 6 regions to place node x or y; (d): wx and vy don't intersect; (e): no two independent nodes in regions 2 and 5

If the disk $D(x,1)$ contains the region $cgdbha = \measuredangle aub - \measuredangle cud$, then clearly node y is inside the disk $D(x,1)$. It implies that xy is an edge in G. Let p and q be the points on line ub such that $\|xp\| = \|xq\| = 1$. Let s be the point on ub such that xs is perpendicular to segment pq and t be the point on ub such that et is perpendicular to segment pq. Clearly, $\|xs\| \leq \|et\|$ since x is inside the triangle $\triangle eub$. It is not difficult to show that $\|ce\| = \|ea\| = 1/2$. Then, $\|et\| = \|ue\| \cdot \sin(\measuredangle aub) < \frac{3}{2}\sin(\frac{\pi}{6}) = 3/4 < \sqrt{3}/2$. It implies that $\measuredangle xqp = \arcsin(\|xt\|/\|xq\|) < \frac{\pi}{3}$. Thus, edge pq is the longest in triangle $\triangle xpq$. Consequently, $\|pq\| > 1$. It is easy to show that, for any two-hop neighbor y of u connected through node v, $\|yv\| \geq \|pq\|$ if both v and y are not inside the disk $D(x,1)$. This is a contradiction to $\|yv\| \leq 1$. This finishes the proof. □

Improved Approximation Algorithms for the Capacitated Multicast Routing Problem

Zhipeng Cai[1,*], Guohui Lin[1,**], and Guoliang Xue[2,***]

[1] Department of Computing Science, University of Alberta
Edmonton, Alberta T6G 2E8, Canada
{zhipeng,ghlin}@cs.ualberta.ca
[2] Department of Computer Science and Engineering, Arizona State University
Tempe, Arizona 85287-5406, USA
xue@asu.edu

Abstract. Two models for the Capacitated Multicast Routing Problem are considered, which are the Multicast k-Path Routing and the Multicast k-Tree Routing. Under these models, two improved approximation algorithms are presented, which have worst case performance ratios of 3 and $(2+\rho)$, respectively. Here ρ denotes the best approximation ratio for the Steiner Minimum Tree problem, and it is about 1.55 at the writing of the paper. The two approximation algorithms improve upon the previous best ones having performance ratios of 4 and $(2.4+\rho)$, respectively. The designing techniques developed in the paper could be applicable to other similar networking problems.

Keywords: Capacitated Multicast Routing, Approximation Algorithm, Steiner Minimum Tree, Tree Partitioning.

1 Introduction

Multicast consists of concurrently sending the same information from a single source node to multiple destination nodes. Multicast service plays an important role in computer and communication networks supporting multimedia applications [8, 10, 14]. It is well known that multicast can be easily implemented on local area networks (LANs) since nodes connected to a LAN usually communicate over a broadcast network. It is also known that implementing multicast in wide area networks (WANs) is quite challenging as nodes connected to a WAN communicate via a switched/routed network [5, 15].

In order to perform multicast communication in WANs, the source node and all the destination nodes must be interconnected. The problem of multicast routing in WANs is thus treated as finding a multicast tree in a network that

* Supported by NSERC.
** To whom correspondence should be addressed. Tel: (780) 492 3737; Fax: (780) 492 1071. Supported by CFI, NSERC, and NNSF Grant 60373012.
*** Supported in part by NSF ITR grant ANI-0312635 and ARO grant W911NF-04-1-0385.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 136–145, 2005.

spans the source and all the destination nodes. Its goal is to minimize the cost of the multicast tree, which is defined to be the sum of the weights of all the edges in the tree.

In this paper, we study the *Capacitated Multicast Routing Problem* in which only a limited number of destination nodes can be assigned to receive the packets sent from the source node during each transmission. Depending on whether or not the switches or routers in the underlying network have the broadcasting ability, two routing models have been considered. In one model, switches are assumed to have the broadcasting ability and the model is called the *multi-tree model* [7]. Multi-tree model has its origin in WDM optical networks with limited light-splitting capabilities. Under this model, we are interested in finding a set of trees such that each tree spans the source node and a limited number of destination nodes that are assigned to receive data and every destination node must be designated to receive data in one of the trees. Compared with the traditional multicast routing model without the capacity constraint – the *Steiner Minimum Tree* problem, which allows any number of receivers in the routing tree, this simpler model makes multicast easier and more efficient to be implemented, at the expense of increasing the cost of the routing tree. Specifically, when the number of destination nodes in a tree is limited to k, we call it the *Multicast k-Tree Routing* (kMTR) problem, which is formally defined in the following.

We model the underlying communication network using a simple, undirected, and edge-weighted graph $G(s, V, D, E)$, where s is the source node, D is the set of n destination nodes, V is the set of all nodes in the network ($s \cup D \subseteq V$), and E is the set of edges (representing communication links) and $w(e) \geq 0$ is the weight (representing the routing cost) of edge $e \in E$. The additive edge weight function $w(\cdot)$ generalizes to subgraphs of G in a natural way. That is, suppose T is a subgraph of G. Then the weight (or cost) of T, denoted by $w(T)$, is the sum of the weights of all the edges in T. Let k be a given positive integer. The multicast k-tree routing (kMTR) problem asks for a partition of D into disjoint sets $D_1, D_2, \ldots, D_\ell$, such that each D_i contains no more than k destination nodes, and a Steiner tree T_i spanning the source node s and the destination nodes in D_i for $i = 1, 2, \ldots, \ell$, such that $\sum_{i=1}^{\ell} w(T_i)$ is minimized.

In the other model, switches have no broadcasting ability and the model is called the *multi-path routing model* [6, 7]. The multi-path model could be viewed as a generalization of one-to-one connection, but a restricted version of the multi-tree routing model. It was proposed for wavelength routed optical networks, and the basic idea is also applicable to general packet switching networks [13]. Under the multi-path model, data is sent from the source node to a destination node in a light path. During the data transmission along the path, if an intermediate node itself is a destination node, then the data is stored (dropped) and a copy of the data is forwarded to its adjacent neighbor down in the path. In each path, some destination nodes are designated where the data is stored (dropped). Accordingly, *multi-path routing* is to find a set of such paths so that every destination node is designated to receive data in one of the paths. Compared with the multi-tree routing model, this simpler model makes multicast easier and more

efficient to be implemented, but again at the expense of increasing the routing tree cost. The parametric variant [6] is the *Multicast k-Path Routing* (kMPR) problem, where every path can be designated with at most k destination nodes.

1.1 State-of-the-Art

For the kMTR problem, the cases where $k = 1, 2$ reduce to the kMPR problem [6]. They both can be solved efficiently. The kMPR problem is NP-hard when $k \geq 3$ [6]. The general case of kMTR, where k is not fixed, is also NP-hard [5]. In [11], kMTR is proven to be NP-hard when k is a fixed integer greater than 2. The best known approximation algorithm for kMPR ($k \geq 3$) has a worst case performance ratio of 4 [6]; The best known approximation algorithm for kMTR ($k \geq 3$) has a worst case performance ratio of $(2.4 + \rho)$ [11], where ρ is the approximation ratio for the Steiner Minimum Tree problem.

1.2 Our Contributions and Organization

We propose a weight averaging technique to facilitate the design and analysis of the better approximation algorithms for both the kMPR and the kMTR problems, for $k \geq 3$. The averaging technique is presented in the next section. Based on it, we give a 3-approximation algorithm for kMPR. We present another technique for partitioning routing trees in Section 3. Combining the tree partitioning technique and the weight averaging technique, we present a $(2+\rho)$-approximation algorithm for kMTR. Here ρ denotes the best approximation ratio for the Steiner Minimum Tree problem, and it is about 1.55 [4, 12] at the writing of this paper. In more detail, we first apply the current best approximation algorithm for the Steiner Minimum Tree problem. Then we partition the obtained approximate Steiner tree to get a number of subtrees each spanning at most k destination nodes, without increasing the total cost. After proving that the sum of the shortest paths from source s to all the subtrees is no more than twice the cost of an optimal k-routing tree, we obtain a $(2 + \rho)$-approximation algorithm for kMTR. This improves upon the previous best approximation ratio of $(2.4 + \rho)$ [11]. We conclude the paper in Section 4.

2 A 3-Approximation Algorithm for kMPR

In the kMPR problem, the underlying communication network can be simplified by removing non-destination nodes since there wouldn't be any Steiner point of degree greater than 2 in a feasible routing tree. It follows that the underlying communication network can be assumed to be an edge-weighted complete graph $G(s, D)$, where s is the source node and $D = \{d_1, d_2, \ldots, d_n\}$ is the destination node set. The weight of an edge is taken to be the cost of the shortest/cheapest path connecting the two ending nodes and thus the edge weight function naturally satisfies triangle inequality. The goal of kMPR is to find a least cost *k-path*

routing, which is a set of paths rooted at s and spanning all the destination nodes, and every path contains at most k destination nodes.

Let $\{P_1^*, P_2^*, \ldots, P_m^*\}$ be the set of paths in an optimal k-path routing. Let $w(P_i^*)$ denote the cost of path P_i^*, which is the sum of the weights of the edges on P_i^*. Let $R^* = \sum_{i=1}^{m} w(P_i^*)$ be the cost of the routing tree.

The 4-approximation algorithm proposed in [6] essentially consists of the following 4 steps: 1) constructing a minimum spanning tree T on $s \cup D$, 2) duplicating the edges in T to produce a Hamiltonian cycle C via suitable short-cutting, 3) partitioning cycle C into segments each containing exactly k distinct destination nodes (the last segment might contain less than k distinct destination nodes), and 4) connecting every segment to source s via a shortest path from s. Since the cost of a minimum spanning tree T is a lower bound for R^* (note that the optimal k-path routing is a spanning tree), the cost of cycle C is no more than $2R^*$. It is shown that the total cost of the shortest-paths, which are added in order to connect the segments to source s, is at most R^*. Since for every segment the shortest path connecting from source s to it could destinate at an internal node on the segment, in order to produce a feasible routing the algorithm uses two copies of the shortest path to generate two paths. Therefore, the cost of the resultant k-path routing could be as large as $4R^*$.

In fact, the following example shows that the ratio 4 is asymptotically tight. In this example, the optimal k-path routing is $\{P_1^*, P_2^*, \ldots, P_m^*\}$, where $P_1^* = s$-d_{mk-1}-d_{mk}-d_1-$\ldots$-d_{k-2}, $P_2^* = s$-d_{k-1}-d_k-d_{k+1}-$\ldots$-d_{2k-2}, $\ldots$, $P_m^* = s$-$d_{(m-1)k-1}$-$d_{(m-1)k}$-$d_{(m-1)k+1}$-$\ldots$-d_{mk-2}. The weights of the edges on the optimal routing are $w(s, d_{ik-1}) = M$ for $i = 1, 2, \ldots, m$, and $w(d_j, d_{j+1}) = 1$ when $j \neq ik - 2$ for some i. The underlying communication network is the completion of the routing tree. Note that the cost of the optimal k-path routing is $R^* = m(M + k - 1)$. The minimum spanning tree has a cost that is the same as the cost of the optimal routing, and the cost of the Hamiltonian cycle is exactly twice R^*. According to the partitioning scheme in the algorithm, $d_1, d_2, \ldots, d_k$ are on a segment and among them d_{k-1} is the closest to source s. Therefore, the output k-path routing by the algorithm has cost $m(4M + 2k - 3)$, which is asymptotically 4 times R^*.

In the following we propose another way to partition the Hamiltonian cycle into segments each containing exactly k distinct destination nodes (again, the last segment might contain less than k distinct destination nodes). For each such segment, we then connect one of its ending destination nodes to source s. To show that this is a 3-approximation algorithm, we will show in the following that the total cost of the added paths is no more than R^*.

For an optimal k-path routing $\{P_1^*, P_2^*, \ldots, P_m^*\}$, in each path P_j^*, the distance from every destination node d_i to source s on the path is an upper bound on the weight of edge (s, d_i) in the underlying network G (recall that we assume the shortest-path distance weight function). Suppose the destination nodes on P_j^* are $d_{j_1}, d_{j_2}, \ldots, d_{j_\ell}$ $(\ell \leq k)$. Then $\sum_{i=1}^{\ell} w(s, d_{j_i}) \leq \ell \times w(P_j^*) \leq k \times w(P_j^*)$. It follows that

$$\sum_{i=1}^{n} w(s, d_i) \leq k \times R^*. \tag{1}$$

Suppose without loss of generality that the destination nodes on the Hamiltonian cycle are indexed consecutively from 1 to n, with source s lying in between d_1 and d_n. Assuming $\ell k < n \leq (\ell+1)k$, make $((\ell+1)k - n)$ copies of s, denote them as $d_{n+1}, \ldots, d_{(\ell+1)k}$, and chain them into a path to replace source s in the Hamiltonian cycle. Note that $w(s, d_i) = 0$, for every $i = n+1, \ldots, (\ell+1)k$; $w(d_n, d_{n+1}) = w(d_n, s)$, $w(d_i, d_{i+1}) = 0$, for $i = n+1, \ldots, (\ell+1)k - 1$, and $w(d_{(\ell+1)k}, d_1) = w(s, d_1)$. In other words, the new Hamiltonian cycle contains exactly $(\ell+1)k$ nodes and its weight is unchanged. Partition the term $\sum_{i=1}^{(\ell+1)k} w(s, d_i)$ into k sub-terms: $\sum_{i=0}^{\ell} w(s, d_{ik+j})$, $j = 1, 2, \ldots, k$. It follows from Equation (1) that there is at least one index j^* such that

$$\sum_{i=0}^{\ell} w(s, d_{ik+j^*}) \leq R^*.$$

Now partition the Hamiltonian cycle into segments of which the first one contains destination nodes d_{j^*}, d_{j^*+1}, d_{j^*+2}, …, d_{j^*+k-1}, the second one contains destination nodes d_{j^*+k}, d_{j^*+k+1}, d_{j^*+k+2}, …, d_{j^*+2k-1}, and so on. The path that is used to connect the i-th segment to source s is edge $(s, d_{(i-1)k+j^*})$. It is clear that the i-th segment appended with edge $(s, d_{(i-1)k+j^*})$ is a path rooted at source s and thus these $(\ell+1)$ paths together form a feasible routing tree. Note that the cost of all the segments is no more than $2R^*$ and the cost of the added edges/paths is no more than R^*. Therefore, the cost of this routing tree has cost no more than $3R^*$.

Theorem 1. *The* kMPR $(k \geq 3)$ *problem admits a* 3*-approximation algorithm that runs in* $O(|D|^3)$ *time.*

Proof. The algorithm presented in the above has a worst case performance ratio of 3. Note that the completion of the graph might take $O(|D|^3)$ time. After that, computing a minimum spanning tree can be done in $O(|D|^2 \log |D|)$ time and forming the Hamiltonian cycle in $O(|D|^2)$ time. It takes $O(|D|)$ time to compute the best partition (or equivalently, the optimal index j^*). Therefore, the overall running time is in $O(|D|^3)$.

3 A $(2+\rho)$-Approximation Algorithm for kMTR

In the kMTR problem, the underlying communication network is a simple, undirected, and edge-weighted complete graph $G(s, V, D)$, where s is the source node, $D = \{d_1, d_2, \ldots, d_n\}$ is the destination node set, V is a superset of D containing also Steiner nodes that can be used as intermediate nodes to save the routing cost, and the edge weight function (the shortest path metric) satisfies triangle inequality. The goal is to find a least cost *k-tree routing*, which is a set of Steiner trees rooted at s and spanning all destination nodes, and every tree contains at most k destination nodes. Note that in a feasible k-tree routing, one destination node assigned in some tree can be used as a Steiner node in others (but not allowed to receive data).

Let $\{T_1^*, T_2^*, \ldots, T_m^*\}$ be the set of trees in an optimal k-tree routing. Let $w(T_i^*)$ denote the cost of tree T_i^*, which is defined to be the sum of the weights of the edges in T_i^*. Let $R^* = \sum_{i=1}^{m} w(T_i^*)$ be the cost of the routing tree. Since every destination node d_i in tree T_j^* satisfies $w(s, d_i) \leq w(T_j^*)$, we have

$$\sum_{i=1}^{n} w(s, d_i) \leq k \times R^*. \tag{2}$$

In the following $(2 + \rho)$-approximation algorithm, we firstly apply the currently best approximation algorithm for the Steiner Minimum Tree problem (which has a worst-case performance ratio of ρ) to obtain a Steiner tree T on $s \cup D$ in the underlying network G. Since the cost of an optimal Steiner tree is a lower bound on R^*, we conclude that the cost of tree T is upper bounded by ρR^*, that is, $w(T) \leq \rho R^*$. Note that tree T is not necessarily a feasible routing tree yet since some branches rooted at source s might contain more than k destination nodes. We process T in the following way: if there is any branch of T (rooted at S) that contains no more than k destination nodes, we can just leave the branch alone in the next step. For those branches each containing more than k destination nodes, we do the following partitioning.

Lemma 1. [11] *Given a Steiner tree T containing $n \geq 3$ destination nodes, it is always possible to partition it into two subtrees that overlap at at most one node, either Steiner or destination, and the number of destination nodes in each subtree falls in the closed interval $[\frac{1}{3}n, \frac{2}{3}n]$.*

Lemma 2. *Given a Steiner tree T containing n destination nodes and an integer k, where $k \geq 3$ and $k < n \leq \frac{3}{2}k$, randomly select $n - \frac{1}{2}k + 1$ destination nodes from the tree to form a set D_0. Then, it is always possible to partition the tree into two subtrees T_1 with destination node set D_1 and T_2 with destination node set D_2, such that T_1 and T_2 overlap at at most one node (either Steiner or destination), $0 < |D_1|, |D_2| \leq k$, $D_1 \cap D_0 \neq \emptyset$, and $D_2 \cap D_0 \neq \emptyset$.*

Proof. Root tree T at any node, which could be either Steiner or destination. In this rooted tree, for every node v, let $c(v)$ denote the number of destination nodes in the subtree rooted at v (inclusive). Let r denote the farthest node from the root that has $c(r) \geq n - \frac{1}{2}k$. Note that r could be the root. Since $k < n \leq \frac{3}{2}k$, r is well-defined and is unique. Re-root tree T at node r. It follows that in the rooted tree, root r is the only node whose c-value is greater than or equal to $n - \frac{1}{2}k$, and consequently its degree is at least 2.

By duplicating root node r, we can partition T into two subtrees (both rooted at r) T_1 with some destination node set D_1 and T_2 with some other destination node set D_2. Note that T_1 and T_2 overlap at root r only. Our partition goal is to satisfy $0 < |D_1|, |D_2| \leq k$, $D_1 \cap D_0 \neq \emptyset$, and $D_2 \cap D_0 \neq \emptyset$. Assuming without loss of generality that $|D_2| \geq |D_1|$. Starting from an arbitrary partition, if our goal is met, then we have obtained two desired subtrees. In the other case, there must be $|D_1| \leq \frac{1}{2}k$ and $|D_2| > n - \frac{1}{2}k$. To prove this, suppose to the contrary that $|D_2| \leq n - \frac{1}{2}k$, or suppose to the contrary that $|D_1| > \frac{1}{2}k$ and consequently

$|D_2| \leq n - \frac{1}{2}k$. We have $|D_1| \leq |D_2| \leq k$. Since $|D_0| = n - \frac{1}{2}k + 1$, we conclude that there must be at least one node from D_0 residing in T_1, and at least one distinct node from D_0 residing in T_2. This is a contradiction.

From $|D_1| \leq \frac{1}{2}k$ and $|D_2| > n - \frac{1}{2}k$, we proceed to examine subtree T_2, which must have multiple branches rooted at r since root r is the only node whose c-value is greater than or equal to $n - \frac{1}{2}k$. For the same reason, each of these branches contains at most $n - \frac{1}{2}k$ destination nodes including root r if r is a destination node. Number these branches as T_{21}, T_{22}, ..., $T_{2\ell}$, and use D_{21}, D_{22}, ..., $D_{2\ell}$ to denote their destination node sets, respectively. We distinguish two cases.

In the first case, there is a branch say T_{2i} such that $|D_{2i}| > \frac{1}{2}k$. It follows from $|D_{2i}| \leq n - \frac{1}{2}k \leq k$ that re-partitioning T to have only T_{2i} in subtree T_2, while all the other branches rooted at r are included into subtree T_1, gives the desired partition. That is, $0 < |D_1|, |D_2| \leq k$, $D_1 \cap D_0 \neq \emptyset$, and $D_2 \cap D_0 \neq \emptyset$. In the second case, every branch contains at most $\frac{1}{2}k$ destination nodes, that is, $|D_{2i}| \leq \frac{1}{2}k$ for $i = 1, 2, \ldots, \ell$. Denote $D_{20} \equiv D_1$ and $T_{20} \equiv T_1$. Since $|D_0| = n - \frac{1}{2}k + 1 > \frac{1}{2}k + 1$, there are at least two branches among $T_{20}, T_{21}, \ldots, T_{2\ell}$, say T_{2i} and T_{2j}, containing distinct destination nodes from D_0. Again, we re-do the partitioning by grouping T_{2i} and some other subtrees as T_1 and grouping T_{2j} and the rest of the subtrees as T_2, to ensure that both T_1 and T_2 contain at most k destination nodes. This can be done since every subtree T_{2i} contains at most $\frac{1}{2}k$ destination nodes. The result is a new pair of subtrees T_1 and T_2 that satisfy $0 < |D_1|, |D_2| \leq k$, $D_1 \cap D_0 \neq \emptyset$, and $D_2 \cap D_0 \neq \emptyset$. This proves the Lemma.

We continue on the partition process. Recall that a branch of T (rooted at source s) containing more than k destination nodes needs to be partitioned. First of all, we delete the edge incident at s from the branch to get a subtree denoted as T_1. Secondly, if T_1 contains more than $\frac{3}{2}k$ destination nodes, we apply Lemma 1 to partition T_1 into two subtrees. We repeatedly apply Lemma 1 to partition every resultant subtree if it contains more than $\frac{3}{2}k$ destination nodes. At the end of this process, there is a set of subtrees of which each contains no more than $\frac{3}{2}k$ destination nodes (and at least $\frac{1}{2}k$ destination nodes since we started with T_1 that contains more than $\frac{3}{2}k$ destination nodes). At this point, for those subtrees that contain no more than k destination nodes, we may leave them alone. For ease of presentation, we call the subtrees containing at most k destination nodes *final trees*. The subtrees that become final at this point are *type-1 final trees*. The non-final subtrees will produce *type-2 final trees* after the next step of partitioning.

For each non-final-yet subtree, again denoted as T_1, our third step is to apply Lemma 2 to partition it into two final subtrees. To this purpose, we let D_0 denote the set of the closest $n_0 - \frac{1}{2}k + 1$ destination nodes (to source s) in T_1, where n_0 is the number of the destination nodes in T_1. Let T_{11} and T_{12} denote the two resultant subtrees having their destination node sets D_1 and D_2, respectively. According to Lemma 2, $0 < |D_1| \leq k$, $D_1 \cap D_0 \neq \emptyset$, $0 < |D_2| \leq k$, and $D_2 \cap D_0 \neq \emptyset$. It is clear that type-2 final trees always come in a pair, since they

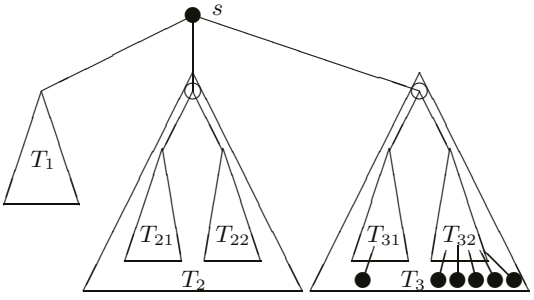


Fig. 1. An illustration of the tree partitioning process: Subtree T_1 contains at most k destination nodes and thus is a final tree; Subtree T_2 contains more than $\frac{3}{2}k$ destination nodes and thus it is partitioned according to Lemma 1; Subtree T_3 contains more than k but at most $\frac{3}{2}k$ destination nodes and thus it is partitioned according to Lemma 2. In the figure, the closest $n_0 - \frac{1}{2}k + 1 = 6$ destination nodes from the source s form set D_0 (shown as filled circles). The resultant subtrees T_{31} and T_{32} both contain some nodes from D_0 (1 node in T_{31} and 5 nodes in T_{32}, respectively).

result from one single partition by Lemma 2. Figure 1 illustrates the schematic partition process.

For each final tree, we identify the closest destination node in the tree and connect it to source s. This gives a feasible k-tree routing. In what follows, we will estimate the total cost of these edges added to connect the final trees to source s. We will show that this total cost is at most twice R^*.

First of all, for every type-1 final tree, we pick the $\frac{1}{2}k$ closest destination nodes in the tree to be the representatives for the tree. Suppose there are ℓ_1 type-1 final trees $T_1, T_2, \ldots, T_{\ell_1}$. Let the representatives for T_i be $d_{i,1}, d_{i,2}, \ldots, d_{i,\frac{k}{2}}$, in the order of non-decreasing distance from source s. Secondly, for every pair of type-2 final trees T_1 and T_2, for each of T_1 and T_2, if it contains no less than $\frac{1}{2}k$ destination nodes, then the $\frac{1}{2}k$ closest ones are picked to be the representatives for the tree; otherwise (i.e., it contains less than $\frac{1}{2}k$ destination nodes), all its destination nodes, say there are m destination nodes, are picked to be the representatives, and additionally the $\frac{1}{2}k - m$ farthest (to source s) destination nodes in its sibling tree are picked to be the representatives. (Note that for its sibling tree, still the $\frac{1}{2}k$ closest destination nodes are picked to be its own representatives.) In this way, every type-2 final tree also has exactly $\frac{1}{2}k$ representatives, which are distinct from any other representatives although some of them might not come from its own but its sibling tree. Note that the reason we can guarantee this property is that the total number of destination nodes in one pair of type-2 final trees is greater than k. Similarly, assume that there are ℓ_2 pairs of type-2 final trees $T_{11}, T_{12}, T_{21}, T_{22}, \ldots, T_{\ell_2 1}, T_{\ell_2 2}$. Let the representatives for T_{ih} be $d_{ih,1}, d_{ih,2}, \ldots, d_{ih,\frac{k}{2}}$, where $h = 1, 2$, in the order of non-decreasing distance from source s. Also for every pair of type-2 final trees T_{i1} and T_{i2}, let $d^0_{i,1}, d^0_{i,2}, \ldots, d^0_{i,\frac{k}{2}}$ be the $\frac{1}{2}k$ closest destination nodes among all the destination nodes in both of them, and let $d^0_{i,\frac{k}{2}+1}, d^0_{i,\frac{k}{2}+2}, \ldots, d^0_{i,k}$ be the $\frac{1}{2}k$

farthest destination nodes among all the destination nodes in both of them. It follows from Equation (2) that

$$\sum_{i=1}^{\ell_1}\sum_{j=1}^{\frac{k}{2}} w(s,d_{i,j}) + \sum_{i=1}^{\ell_2}\sum_{j=1}^{k} w(s,d^0_{i,j}) \leq \sum_{i=1}^{n} w(s,d_i) \leq k \times R^*.$$

Using the non-decreasing distance orderings of these destination nodes, we have

$$\sum_{i=1}^{\ell_1} w(s,d_{i,1}) + \sum_{i=1}^{\ell_2} \left(w(s,d^0_{i,1}) + w(s,d^0_{i,\frac{k}{2}+1}) \right) \leq 2R^*.$$

Clearly, for every type-1 final tree T_i, destination node $d_{i,1}$ is connected to source s; also is true that $d^0_{i,1}$ must serve as a representative for either type-2 final tree T_{i1} or T_{i2} and thus it is connected to source s. Suppose without loss of generality that $d^0_{i,1}$ is a representative for T_{i1}, then the closest destination node $d_{i2,1}$ in T_{i2}, which is picked to be a representative, has a distance no larger than the distance from source to destination node $d^0_{i,\frac{k}{2}+1}$. That is, $w(s,d_{i2,1}) \leq w(s,d^0_{i,\frac{k}{2}+1})$. It follows that the total cost of the edges added to connect the source to the final trees to produce a feasible k-tree routing is at most $2R^*$. Therefore, the routing tree produced in this way has cost no more than $(2+\rho)R^*$.

INPUT: an edge-weighted graph $G(s,V,D)$;
OUTPUT: a k-tree routing.
1. Compute a Steiner tree T_0 on $s \cup D$, using the currently best approximation;
2. Delete the edges incident at s to obtain subtrees of T_0;
3. For each subtree T_1 containing $k' > k$ destination nodes:
3.1 if $k' > \frac{3}{2}k$, apply Lemma 1 to partition T_1;
3.2 if $k' \leq \frac{3}{2}k$, apply Lemma 2 to partition T_1;
4. For every final tree:
4.1 connect the closest destination node in the tree to source s;
5. Output the k-tree routing.

Fig. 2. A high-level description of the $(2+\rho)$-approximation algorithm for kMTR, where $k \geq 3$.

Theorem 2. kMTR ($k \geq 3$) *admits a* $(2+\rho)$*-approximation algorithm, where* ρ *is the best performance ratio for approximating the* Steiner Minimum Tree *problem.*

Proof. The algorithm presented in the above, with its high-level description in Figure 2, has been proven to be a $(2+\rho)$-approximation for the kMTR problem. It is worth mentioning that the running time of the algorithm is dominated by the best approximation for the Steiner Minimum Tree problem.

Acknowledgment

The authors would like to thank a referee for pointing out reference [9] in which independently another $(2+\rho)$-approximation algorithm is designed for the kMTR problem. Furthermore, they realized that an algorithm in [1] is a $(2\rho+1)$-approximation algorithm. They also want to remark that the $(2.4+\rho)$-approximation algorithm in [11] was designed in year 2003 and the $(2+\rho)$-approximation algorithm presented in the above was designed before [9] was made public (see a master thesis [2] and technical report [3]).

References

1. K. Altinkemer and B. Gavish. Heuristics with constant error guarantees for the design of tree networks. *Management Science*, 34:331–341, 1988.
2. Z. Cai. Improved algorithm for multicast routing and binary fingerprint vector clustering. Master's thesis, Department of Computing Science, University of Alberta, June 16, 2004.
3. Z. Cai, G.-H. Lin, and G. L. Xue. Improved approximation algorithms for the capacitated multicast routing problem. Technical Report TR04-12, Department of Computing Science, University of Alberta, July 2004.
4. C. Gröpl, S. Hougardy, T. Nierhoff, and H. J. Prömel. Approximation algorithms for the Steiner tree problem in graphs. In D.-Z. Du and X. Cheng, editors, *Steiner Trees in Industries*, pages 235–279. Kluwer Academic Publishers, 2001.
5. J. Gu, X. D. Hu, X. Jia, and M.-H. Zhang. Routing algorithm for multicast under multi-tree model in optical networks. *Theoretical Computer Science*, 314:293–301, 2004.
6. J. Gu, X. D. Hu, and M.-H. Zhang. Algorithms for multicast connection under multi-path routing model. *Information Processing Letters*, 84:31–39, 2002.
7. R. L. Hadas. Efficient collective communication in WDM networks. In *Proceedings of IEEE ICCCN 2000*, pages 612–616, 2000.
8. C. Huitema. *Routing in the Internet*. Prentice Hall PTR, 2000.
9. R. Jothi and B. Raghavachari. Approximation algorithms for the capacitated minimum spanning tree problem and its variants in network design. In *Proceedings of ICALP 2004*, LNCS 3142, pages 805–818, 2004.
10. F. Kuo, W. Effelsberg, and J. J. Garcia-Luna-Aceves. *Multimedia Communications: Protocols and Applications*. Prentice Hall, Inc., 1998.
11. G.-H. Lin. An improved approximation algorithm for multicast k-tree routing. *Journal of Combinatorial Optimization*, 2005. In press (final version available upon request).
12. G. Robins and A. Z. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the* 11*th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pages 770–779, 2000.
13. A. S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, Upper Saddle River, NJ, 1996.
14. Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14:1228–1234, 1996.
15. X. Zhang, J. Wei, and C. Qiao. Constrained multicast routing in WDM networks with sparse light splitting. In *Proceedings of IEEE INFOCOM 2000*, pages 1781–1790, March 26–30, 2000.

Construction of Scale-Free Networks with Partial Information

Jianyang Zeng, Wen-Jing Hsu*, and Suiping Zhou

Center for Advanced Information Systems, Nanyang Technological University
Singapore 639798
zengjy@gmail.com, {hsu,asspzhou}@ntu.edu.sg

Abstract. It has recently been observed that the node degrees of many real-world large-scale networks, such as the Internet and the Web, follow a power law distributions. Since the classical random graph models are inadequate for explaining this phenomenon, alternative models have been proposed. However, most of the existing models unrealistically assume that each new joining node knows about all the existing nodes in the network. We relax this assumption and propose a model in which each new joining node uniformly and randomly chooses a sample set of existing nodes, and then connects to some nodes in the sample set according to the Preferential Attachment rule. We show that the power law of degree distribution still holds true even if each new joining node knows only a logarithmic number of existing nodes. Compared with the existing models, our construction of scale-free networks based on partial information seems to better approximate the evolution of certain complex networks arising in the real world. Our results may also be applied to the constructions of large-scale distributed systems such as peer-to-peer networks, where the global information is generally unavailable for any individual node in the network.

1 Introduction

It has been reported empirically by several researchers, e.g. in [2, 11, 17], that both the Internet and the Web have a *scale-free* (i.e. size-independent) property: The proportion P_k of nodes with degree k follows a power law distribution: $P_k \sim k^{-r}$, where $r \leq 3$. The well known models of random graphs introduced by Erdős and Rényi [21] do not yield the power law distribution, and thus require modifications for modelling and analyzing these large-scale networks. Barabási and Albert [4] proposed the first scale-free network model referred to as the BA model based on the preferential attachment (PA) rule. The BA model is a dynamic stochastic process, i.e., a new node is added into the network at each time step, and the probability that an existing *old* node gets a link from the new node is proportional to its degree. Kumar et al. [19] independently presented a so-called *copying model*, motivated by the fact that a new web node is often generated by copying an old one and changing some of its old links. Aiello, Chung,

* Corresponding author.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 146–155, 2005.

and Lu [1] introduced a scale-free model with a prescribed degree sequence, where the probability of two nodes being connected is proportional to the product of their expected degrees. Fabrikant, Koutsoupias, and Papadimitriou (FKP) [16] considered the underlying geometric metric and proposed a "Highly Optimized Tolerance" (HOT) framework to model the Internet network, i.e., the connection of the new node is constructed according to an adjustable neighborhood consideration. The more details of degree distribution of the FKP model and its extended model are studied in [5] and [6] respectively. A mathematical survey on some scale-free models can be found in [7].

Many variations of the preferential attachment rule originated from the BA model have been developed [9, 10, 12–15]. Bollobás and Riordan refined the BA model and make it more precise and theoretically analyzable [9, 10]. They found that although the BA model is a dynamic stochastic process, it can be regarded as a "LCD (Linearized Chord Diagrams)" [9], which is a static problem and is easier to solve. A variation on the BA model, in which each node has an "initial attractiveness" was introduced independently by two different groups, Dorogovtsev et al. [14] and Drinea et al. [15]. A precise and rigorous analysis of this model was given by Buckley and Osthus [12]. Cooper and Frieze [13] presented a general model which considers more parameters, such as a variable number of edges or nodes generated at each time step, a mixture of uniform and PA selection, etc.

Although these scale-free models are consistent with the power law observation of real-world large-scale networks, unfortunately, most of existing models assume that a newly added node knows all existing nodes, such as their node degrees [4], or the Euclidean distance [16], or the hop distance to the network center [6], etc. This is an unrealistic assumption with real-world large-scale networks such as the Internet or Web, as it requires a node to access and process the extremely large amount of the global information. We will, instead, assume that each node has access to only a small subset of logarithmic size of all the existing nodes.

As in [4, 9], we also allow only one node to be added into the network at each time step. The node uniformly and randomly chooses a sample set of existing nodes, and then connects to some nodes in the sample set according to the PA rule. Our results show that the power law of degree distribution still holds true even if each new joining node knows only a logarithmic number of existing nodes. Compared to the existing models, our construction of scale-free networks based on partial information seems to better approximate the evolution of real-world complex networks.

Our results may also be applied to the constructions of large-scale distributed systems such as peer-to-peer networks, where the global information is generally unavailable for any individual node in the network.

1.1 The Model and Notations

Let m denote an integer constant. Our construction is described as follows.

Step 1:
Start with G_1^m, the graph with only one single node denoted by v_1 with m self-loops.
Step *t*:
A new node denoted by v_t is added to the graph G_{t-1}^m to form G_t^m. This new node sends m edges to the existing nodes in G_{t-1}^m according to the following rule: First randomly and independently choose S_{t-1} nodes from G_{t-1}^m to form the sample set T_{t-1}. Then node v_t sends m edges to the nodes in the sample set T_{t-1} according to the preferential attachment rule, i.e., the probability that v_t is connected to a node $u \in T_{t-1}$ is

$$Pr[v_t \to u] = \frac{deg_{t-1}(u)}{\sum_{i \in T_{t-1}} deg_{t-1}(i)},$$

where $deg_{t-1}(x)$ denotes the degree of node x in the graph G_{t-1}^m.

We will show that to ensure the power law, it suffices to choose $S_t = \beta \lg t$ [1], where β denotes a constant to be specified later. During the initial steps, it is possible that $S_t \leq t$. In this case, we choose all the existing nodes as the sample set. Such initial choices will not affect our asymptotic results. We assume that all edges in the graph are undirected. The directed variant of our model can be easily obtained by applying a similar method as in [13].

Below are the notations that will be used in our analysis.

$deg_t(x)$: The degree of node x in the graph G_t^m;
T_t: The sample set chosen at the time step t;
S_t: The size of the sample set T_t, i.e., $S_t = |T_t|$;
D_t: The sum of the node degrees of all nodes in the sample set T_t, i.e., $D_t = \sum_{x \in T_t} deg_t(x)$;
$d_t(i)$: The number of nodes with degree i in the graph G_t^m;
$d_{t,s}(i)$: The number of nodes with degree i in the sample set T_t;
$\bar{d}_t(i)$: The expectation of $d_t(i)$;
$\bar{d}_{t,s}(i)$: The expectation of $d_{t,s}(i)$;
M_t: The maximum degree of the graph G_t^m;
Diam(G_t^m): The diameter of the graph G_t^m;
n: The size of the final graph G_n^m.

Throughout the paper, we will identify the node v_x with the integer number x to simplify the notation.

1.2 Our Main Results

Our main results are stated as follows:

[1] The logarithmic symbol log is with the base 2, if not otherwise specified. Also, we remove the ceiling or floor for simplicity throughout the paper.

Result 1 (Node Degree Distribution): *Let m and β denote sufficiently large constants, and let $S_t = \beta \lg t$ denote the size of the sample set in Step t. Let $d_n(i)$ denote the number of nodes with degree i in G_n^m. Then* **whp**

$$d_n(i) = \frac{cn}{i(i+1)(i+2)} + O(n^{\theta}),$$

where θ and c denote constants, and $0 < \theta < 1$.

Since each node has a degree of at least m, it is trivial to calculate $d_t(i)$ for $i \leq m$. In the subsequent analysis, when we talk about $d_t(i)$, we assume that $i > m$.

Result 2 (Maximum Degree): *Let M_n denote the maximum degree of the graph G_n^m. Then for any constant $\epsilon \in (0, \frac{1}{2})$,* **whp**

$$C_2 n^{\frac{1}{2}-\epsilon} \leq M_n \leq C_1 n^{\frac{1}{2}+\epsilon},$$

where C_1 and C_2 denote constants.

Result 3 (Network Diameter): *Let $\mathrm{Diam}(G_n^m)$ denote the diameter of the graph G_n^m. If $m \geq C_3 \lg n$ for a sufficiently large constant C_3, then* **whp**

$$\mathrm{Diam}(G_n^m) \leq 2 \lg n.$$

Although we only show that $\mathrm{Diam}(G_n^m) = O(\lg n)$ for the case $m = \Omega(\lg n)$, we conjecture that $\mathrm{Diam}(G_n^m) = O(\lg n)$ also holds for $m = \Theta(1)$.

The proof of **Result 1** is relatively more challenging and its proof is given in Section 2. Due to page limitation, the proofs of **Result 2** and **Result 3** are not shown in this conference paper. The reader is referred to [22] for more details.

2 The Degree Distribution

The following is a roadmap for the proof of **Result 1**. Firstly, Section 2.1 gives the expectation of D_t and shows that D_t concentrates around its expectation by applying an extension of the martingale method [20]. Secondly, based on the concentration result of D_t, a recurrence relation of $\bar{d}_t(i)$ is given in Section 2.2. We then inductively show that its solution follows a power law. In Section 2.3, we argue that $d_t(i)$ concentrates around its expectation by applying a similar analysis of the concentration of D_t.

2.1 Concentration of D_t

We first analyze the expectation of D_t.

Since the sample nodes in T_t are selected randomly and independently from the current nodes in G_t^m, we can easily obtain the following lemma.

Lemma 1. $E[d_{t,s}(i)] = \frac{S_t}{t} E[d_t(i)]$.

Based on the above lemma, the expectation of D_t can be easily obtained.

Lemma 2. $E[D_t] = 2mS_t$.

Proof:

$$E[D_t] = E[\sum_{i>0} d_{t,s}(i)] = \sum_{i>0} E[d_{t,s}(i)] = \sum_{i>0} \frac{S_t}{t} E[d_t(i)] \quad \text{(by Lemma 1)}$$
$$= \frac{S_t}{t} \sum_{i>0} E[d_t(i)] = \frac{S_t}{t} E[\sum_{i>0} d_t(i)] = \frac{S_t}{t} 2mt = 2mS_t. \quad \blacksquare$$

Our analysis of the concentration result is mainly based on the following probabilistic tool, which is an extension of the martingale method [3].

Lemma 3 (Martingale extension [20]). *Let $X = (X_1, \cdots, X_n)$ be a family of random variables with X_k taking value in a set A_k, and let f be a bounded real-valued function defined on $\Pi A_k = A_1 \times A_2 \times \cdots \times A_n$. Let $x_i \in A_i$ for each $i = 1, \cdots, k-1$. For $x \in A_k$, let $g_k(x) = E[f(X)|X_k = x] - E[f(X)]$. Let $r_k = \sup\{|g_k(x) - g_k(y)| : x, y \in A_k\}$, and $R^2(X) = \sum_{k=1}^{n} r_k^2$. Let $\hat{r}^2 = \sup\{R^2(X) for\ all\ X \in \Pi A_k\}$. Then*

$$\Pr[|f(X) - E[f(X)]| \geq c] \leq 2\exp(-2c^2/\hat{r}^2),$$

where $c > 0$

The following lemma gives a lower bound of D_t which is useful for our subsequent analysis. Due to space limitation, its proof is referred to our full version [22].

Lemma 4. *Let m and β denote sufficiently large constants, and let $S_t = \beta \lg t$ as defined earlier and let $n_0 = n^{5/6}$. Then there exists a constant δ where $0 < \delta < 1$, such that*

$$\Pr\left[D_t \geq (1+\delta)mS_t\right] > 1 - \frac{1}{n^2},$$

for all $t > n_0$.

Let N_i denote the set of neighbors of node i when it first enters the network at step i. Then N_i is a tuple of nodes $\boldsymbol{x} = (x_1, \cdots, x_m) \in \{1, \cdots, i-1\}^m$. Let $g_{\tau,t}(\boldsymbol{x}) = E[D_t|N_1, \cdots, N_{\tau-1}, N_\tau = \boldsymbol{x}]$, where the sequence of $N_1, \cdots, N_{\tau-1}$ is fixed and $\boldsymbol{x} \in \{1, \cdots, \tau-1\}^m$, $1 \leq \tau \leq t$. Let $r_{\tau,t} = \sup\left\{|g_{\tau,t}(\boldsymbol{x}) - g_{\tau,t}(\boldsymbol{y})| : \boldsymbol{x}, \boldsymbol{y} \in \{1, \cdots, \tau-1\}^m\right\}$. In order to bound $r_{\tau,t}$ and apply the martingale extension in Lemma 3 to analyze the concentration of D_t, we introduce the following node-edge marking rule:

As in [9, 10], we regard one edge as two "half-ward" directed edges. At a time step τ, we initially mark the nodes in the set N_τ as "τ-influenced nodes", and mark all the half edges connected to N_τ as "τ-influenced half edges". During the next time step $\tau + 1$, if an outgoing half edge sent by the new node $\tau + 1$

is connected to a τ-influenced node, it is also marked as a τ-influenced edge. Applying the same rule for the constructions of the sequence of graphs $G^m_{\tau+2}, \cdots, G^m_t$, then the value of $E[D_t|N_1, \cdots, N_{\tau-1}, N_\tau = \boldsymbol{x}] - E[D_t|N_1, \cdots, N_{\tau-1}, N_\tau = \boldsymbol{y}]$ is upper bounded by the number of all τ-influenced edges that are attached to the sample set T_t. Let Δ_σ denote the expected number of τ-influenced edges in the graph G^m_σ, then we have $r_{\tau,t} \leq \frac{S_t}{t}\Delta_t$.

Based on the above observations, we have the following lemma.

Lemma 5. *Define $r_{\tau,t}$ and Δ_τ as above. Let m and β denote sufficiently large constants, and let $S_t = \beta \lg t$. Let δ denote a constant such that $0 < \delta < 1$. Then,*

$$r_{\tau,t} \leq \begin{cases} \frac{S_t}{t}\Delta_\tau \frac{n_0}{\tau}\left(\frac{t}{n_0}\right)^{\frac{1}{1+\delta}}, & \text{when } 1 \leq \tau \leq n_0; \\ \frac{S_t}{t}\Delta_\tau \left(\frac{t}{\tau}\right)^{\frac{1}{1+\delta}}, & \text{when } n_0 < \tau \leq t. \end{cases}$$

where $n_0 = n^{5/6}$.

Proof: Recall that Δ_σ is the expected number of τ-influenced edges in the graph G^m_σ. Let Y_σ denote the expected number of new τ-influenced edges generated from G^m_σ during step $\sigma + 1$. Then by linearity of expectation, we have $\Delta_{\sigma+1} = \Delta_\sigma + Y_\sigma$.

Let e^σ_i denote the number of nodes with i τ-influenced edges among G^m_σ, then $\Delta_\sigma = \sum_{i>0} i \cdot e^\sigma_i$. So

$$Y_\sigma \leq \sum_{i>0} \frac{m \cdot e^\sigma_i \cdot \frac{S_\sigma}{\sigma} \cdot i}{D_\sigma} = \frac{mS_\sigma}{\sigma D_\sigma} \sum_{i>0} e^\sigma_i \cdot i = \frac{mS_\sigma \Delta_\sigma}{\sigma D_\sigma}.$$

We bound Y_σ according to two different cases.

Case 1 $n_0 < \sigma < t$**:** Let $\mathcal{E}_\sigma$ denote the event that $D_\sigma \geq (1+\delta)mS_\sigma$. From Lemma 4, we have $\Pr[\mathcal{E}_\sigma] \geq 1 - \frac{1}{n^2}$. Let $\mathcal{E} = \bigcap_{\sigma=\tau}^t \mathcal{E}_\sigma$, then the event $\mathcal{E}$ occurs with probability at least $1 - \frac{1}{n}$. Thus we can assume that $D_\sigma \geq (1+\delta)mS_\sigma$ for all $n_0 < \sigma \leq t$ in the following. Thus,

$$Y_\sigma \leq \frac{mS_\sigma \Delta_\sigma}{\sigma D_\sigma} \leq \frac{\Delta_\sigma}{(1+\delta)\sigma}$$

Case 2 $\tau < \sigma \leq n_0$**:** It is obvious that $D_\sigma \geq mS_\sigma$. So we have

$$Y_\sigma \leq \frac{mS_\sigma \Delta_\sigma}{\sigma D_\sigma} \leq \frac{\Delta_\sigma}{\sigma}$$

Combining the above two cases, when $1 \leq \tau \leq n_0$, we have

$$\Delta_t \leq \Delta_\tau \prod_{\sigma=\tau+1}^{n_0} \frac{\sigma+1}{\sigma} \prod_{\sigma=n_0+1}^{t} \left(1 + \frac{1}{(1+\delta)\sigma}\right) \leq \Delta_\tau \frac{n_0}{\tau}\left(\frac{t}{n_0}\right)^{\frac{1}{1+\delta}}$$

By using the fact that $1 + ax \leq (1 + x^a)$ for $x > -1$ and $a \geq 1$, we have $1 + \frac{1}{(1+\delta)\sigma} \leq (1 + \frac{1}{\sigma})^{1/(1+\delta)}$. Thus, we have

$$\Delta_t \leq \Delta_\tau \prod_{\sigma=\tau+1}^{n_0} \frac{\sigma+1}{\sigma} \prod_{\sigma=n_0+1}^{t} \left(1 + \frac{1}{\sigma}\right)^{\frac{1}{1+\delta}} \leq \Delta_\tau \frac{n_0}{\tau}\left(\frac{t}{n_0}\right)^{\frac{1}{1+\delta}}$$

When $n_0 < \tau < t$, only case 1 applies, so we have

$$\Delta_t \leq \Delta_\tau \prod_{\sigma=\tau+1}^{t} \big(1+\frac{1}{(1+\delta)\sigma}\big) \leq \Delta_\tau \prod_{\sigma=\tau+1}^{t} \big(1+\frac{1}{\sigma}\big)^{\frac{1}{1+\delta}} \leq \Delta_\tau \big(\frac{t}{\tau}\big)^{\frac{1}{1+\delta}}$$

Since $r_{\tau,t} \leq \frac{S_t}{t}\Delta_t$, we have

$$r_{\tau,t} \leq \begin{cases} \frac{S_t}{t}\Delta_\tau \frac{n_0}{\tau}\big(\frac{t}{n_0}\big)^{\frac{1}{1+\delta}}, & \text{when } 1 \leq \tau \leq n_0; \\ \frac{S_t}{t}\Delta_\tau \big(\frac{t}{\tau}\big)^{\frac{1}{1+\delta}}, & \text{when } n_0 < \tau \leq t. \end{cases}$$

■

Theorem 1. *Let m and β denote sufficiently large constants, and let $S_t = \beta \lg t$ as before. Let $n_1 = n^{\frac{11}{12}}$. Then there exists a constant $0 < \varphi < 1$ such that*

$$\Pr[|D_t - 2mS_t| \geq mS_t t^{\varphi-1}] \leq \frac{1}{n^2},$$

for all $t > n_1$.

Proof: Let $R_t^2 = \sum_{\tau=1}^{t}(r_{\tau,t})^2$. From Lemma 5, we have

$$\begin{aligned} R_t^2 = \sum_{\tau=1}^{t}(r_{\tau,t})^2 &\leq \sum_{\tau=1}^{n_0}\big(\frac{S_t}{t}\Delta_\tau\frac{n_0}{\tau}(\frac{t}{n_0})^{\frac{1}{1+\delta}}\big)^2 + \sum_{\tau=n_0+1}^{t}\big(\frac{S_t}{t}\Delta_\tau(\frac{t}{\tau})^{\frac{1}{1+\delta}}\big)^2 \\ &= \big(\frac{S_t}{t}\Delta_\tau t^{\frac{1}{1+\delta}}\big)^2\big(n_0^{2-\frac{2}{1+\delta}}\sum_{\tau=1}^{n_0}\frac{1}{\tau^2} + \sum_{\tau=n_0+1}^{t}\frac{1}{\tau^2}\big) \\ &= O\big((\frac{S_t}{t}\Delta_\tau t^{\frac{1}{1+\delta}} n_0^{1-\frac{1}{1+\delta}})^2\big) \end{aligned}$$

Since node τ affects at most $2m$ degrees at the time step τ, $\Delta_\tau \leq 2m$. Hence, $R_t^2 = O\big((\frac{S_t}{t}\Delta_\tau t^{\frac{1}{1+\delta}} n_0^{1-\frac{1}{1+\delta}})^2\big) = O\big((\frac{S_t}{t} m t^{\frac{1}{1+\delta}} n_0^{1-\frac{1}{1+\delta}})^2\big)$.

So $\hat{r}^2 = \sup\{R_t^2\} = O\big((\frac{S_t}{t} m t^{\frac{1}{1+\delta}} n_0^{1-\frac{1}{1+\delta}})^2\big)$. By Lemma 3, we have

$$\Pr\big[|D_t - 2mS_t| \geq \frac{mS_t}{t} t^{\frac{1}{1+\delta}} n_0^{1-\frac{1}{1+\delta}} \lg n\big] = \exp\big(-\Omega(\lg^2 n)\big)$$

Since $n_1 = n^{\frac{11}{12}} > n_0 = n^{\frac{5}{6}}$, there exists a constant $0 < \varphi < 1$ such that

$$\Pr\big[|D_t - 2mS_t| \geq mS_t t^{\varphi-1}\big] \leq \frac{1}{n^2},$$

for all $t > n_1$ ■

2.2 Power Law Distribution of $\bar{d}_t(i)$

Theorem 2. *Let m and β denote sufficiently large constants, and let $S_t = \beta \lg t$. Then there exists a constant $0 < \theta < 1$ such that* **whp**

$$\bar{d}_n(i) = \frac{cn}{i(i+1)(i+2)} + O(n^\theta),$$

for a constant c.

Proof: By construction of our model (cf. Section 1.1), we have the following relation:

$$E[d_{t+1}(i)|G_t^m] = d_t(i) + md_{t,s}(i-1)\frac{i-1}{D_t} - md_{t,s}(i)\frac{i}{D_t}.$$

Taking the expectation on both sides, we have

$$E[d_{t+1}(i)] = E[d_t(i)] + m(i-1)E\big[\frac{d_{t,s}(i-1)}{D_t}\big] - miE\big[\frac{d_{t,s}(i)}{D_t}\big]. \tag{1}$$

Let $\mathcal{F}$ denote the event that $|D_t - 2mS_t| < mS_t t^{\varphi-1}$, then $\Pr[\neg\mathcal{F}] \le n^{-2}$ for all $t > n_1 = n^{\frac{11}{12}}$ according to Theorem 1. It is obvious that $D_t \ge i \cdot d_{t,s}(i)$, hence

$$\begin{aligned} E\big[\frac{d_{t,s}(i)}{D_t}\big] &= E\big[\frac{d_{t,s}(i)}{D_t}|\mathcal{F}\big]\Pr[\mathcal{F}] + E\big[\frac{d_{t,s}(i)}{D_t}|\neg\mathcal{F}\big]\Pr[\neg\mathcal{F}] \\ &\le E\big[\frac{d_{t,s}(i)}{D_t}|\mathcal{F}\big]\Pr[\mathcal{F}] + \frac{\Pr[\neg\mathcal{F}]}{i}. \end{aligned}$$

Let $a = 2mS_t$ and $b = mS_t t^{\varphi-1}$. In the event $\mathcal{F}$, we have $a - b \le D_t \le a + b$. Thus, $\frac{1}{D_t} \le \frac{1}{a-b}$ in this case. So

$$\begin{aligned} &\frac{1}{D_t}\big(\frac{a-b}{a}\big) \le \big(\frac{1}{a-b}\big)\big(\frac{a-b}{a}\big) = \frac{1}{a} \\ \Rightarrow\ &\frac{1}{D_t} \le \frac{1}{a} + \frac{b}{a}\frac{1}{D_t} = \frac{1}{2mS_t} + \frac{t^{\varphi-1}}{2}\frac{1}{D_t} \end{aligned}$$

Thus we have

$$\begin{aligned} E\big[\frac{d_{t,s}(i)}{D_t}\big] &\le E\big[\frac{d_{t,s}(i)}{2mS_t} + \frac{d_{t,s}(i)}{D_t}\frac{t^{\varphi-1}}{2}\big]\Pr[\mathcal{F}] + \frac{\Pr[\neg\mathcal{F}]}{i} \\ &\le \frac{E[d_{t,s}(i)]}{2mS_t} + \frac{t^{\varphi-1}}{2i} + \frac{\Pr[\neg\mathcal{F}]}{i}. \end{aligned}$$

Since $\Pr[\neg\mathcal{F}] \le n^{-2}$ for all $t > n_1 = n^{\frac{11}{12}}$, we have

$$E\big[\frac{d_{t,s}(i)}{D_t}\big] \le \frac{E[d_{t,s}(i)]}{2mS_t} + \frac{t^{\varphi-1}}{i}.$$

According to Lemma 1, $E[d_{t,s}(i)] = \frac{S_t}{t}E[d_t(i)]$. So if $t > n_1 = n^{\frac{11}{12}}$, we have **whp**

$$E\big[\frac{d_{t,s}(i)}{D_t}\big] = \frac{\bar{d}_t(i)}{2mt} + O\big(\frac{t^{\varphi-1}}{i}\big).$$

Similarly, we obtain

$$E\big[\frac{d_{t,s}(i-1)}{D_t}\big] = \frac{\bar{d}_t(i-1)}{2mt} + O\big(\frac{t^{\varphi-1}}{i-1}\big).$$

Thus, Eq. (1) can be converted into

$$\bar{d}_{t+1}(i) = \bar{d}_t(i) + \frac{\bar{d}_t(i-1)}{2t}(i-1) - \frac{\bar{d}_t(i)}{2t}i + O\big(t^{\varphi-1}\big),$$

for all $t > n_1 = n^{\frac{11}{12}}$.

This formula is similar to the recurrence equation in [18]. Thus, by a similar inductive analysis, we can get the following solution for all $1 \leq t \leq n$:

$$\bar{d}_t(i) = \frac{ct}{i(i+1)(i+2)} + O(n^{\theta}),$$

where c denotes a constant, and $\max\{\varphi, \frac{11}{12}\} < \theta < 1$. ∎

2.3 Concentration of $d_t(i)$

Theorem 3. *Let m and β denote sufficiently large constants, and let $S_t = \beta \lg t$ for our model. Then there exists a constant $0 < \xi < 1$ such that*

$$\Pr\left[|d_n(i) - \bar{d}_n(i)| \geq n^{\xi}\right] \leq n^{-2}.$$

Proof: The proof can be obtained by applying a similar analysis of the concentration of D_t. ∎

3 Concluding Remarks

We have proposed a new scale-free model for large-scale networks where a new joining node connects to some nodes in a small sample set; the connections follow the preferential attachment rule. We show that the power law of degree distribution still holds true. Compared with the existing models, our construction based on partial information can better approximate the evolution of large-scale distributed systems arising in the real world. Our results may also be applied to the constructions of peer-to-peer networks, where the global information is generally unavailable for any individual node in the network.

We have also experimentally evaluated the distribution of node degree in our model. Due to space limitation, the reader is referred to [22] for more details. Our simulations show that, when $|S_t| \geq 5$, the proportion P_k of nodes with degree k follows a power law distribution: $P_k \backsim k^{-r}$, where $r \approx 3$ denotes as the slope of the log-log curve. Our experimental results indicate that our theoretical result (Result 1) is a little conservative. We conjecture that when $|S_t| = \Omega(1)$, the distribution will obey a power law distribution. The rigorous proof of this conjecture remains open.

Acknowledgement

We wish to thank Abraham D. Flaxman for useful discussions over emails during the preparation of this work. We would also like to thank anonymous referees for their numerous remarks.

References

1. W. Aiello, F.R.K. Chung, and L. Lu. A Random Graph Model for Massive Graphs. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, 2000.

2. R. Albert, H. Jeong, and A.-L. Barabási. The Diameter of the World Wide Web. *Nature*, 401(9):130–131, 1999.
3. N. Alon and J.H. Spencer. *The Probabilistic Method*. John Wiley, 2000.
4. A.-L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286:509–512, 1999.
5. N. Berger, B. Bollobás, C. Borgs, J. Chayes, and O. Riordan. Degree Distribution of the FKP Network Model. In *Proceedings of the 13th International Colloquium on Automata, Languages and Programming*, 2003.
6. N. Berger, C. Borgs, J.T. Chayes, R.M. D'Souza, and R.D. Kleinberg. Degree Distribution of Competition-Induced Preferential Attachment Graphs. In *Proceedings of the 14th International Colloquium on Automata, Languages and Programming*, 2004.
7. B. Bollobás and O. Riordan. *Mathematical Results on Scale-Free Random Graphs*. In Handbook of Graphs and Neworks, 2002.
8. B. Bollobás and O. Riordan. Coupling Scale-Free and Classical Random Graphs. *Internet Mathematics*, 1(2):215–225, 2003.
9. B. Bollobás and O. Riordan. The Diameter of a Scale-Free Random Graph. *Combinatorica*, to appear.
10. B. Bollobás, O. Riordan, J. Spencer, and G. Tusnády. The Degree of Sequence of a Scale-Free Random Graph Process. *Random Structures and Algorithms*, 18:279–290, 2001.
11. A. Broder, R. Kumar, F. Maghoul, P. Raghavan S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph Structure in the Web. In *Proceedings of the 9th International World Wide Web Conference*, 2002.
12. G. Buckley and D. Osthus. Popularity Based Random Graph Models Leading to a Scale-Free Degree Distribution, Submitted. 2001.
13. C. Cooper and A. Frieze. A General Model of Web Graphs. *Random Structures and Algorithms*, 22:311–335, 2003.
14. S.N. Dorogovtsev, J.F.F. Mendes, and A.N. Samukhin. Sturcture of Growing Networks with Preferential Linking. *Physical Review Letters*, 85(21):4633–4636, 2000.
15. E. Drinea and M. Enachescu and M. Mitzenmacher. Variations on Random Graph Models for the Web. Technical report, Department of Computer Science, Harvard University, 2001.
16. A. Fabrikant, E. Koutsoupias, and C. Papadimitriou. Heuristically Optimized Trade-Offs: a New Paradigm for Power Laws in the Internet. In *Proceedings of the 29th International Colloquium on Automata,Languages and Programming*, 2002.
17. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proceedings of ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 1999.
18. A.D. Flaxman, A.M. Frieze, and J. Vera. A Geometric Preferential Attachment Model of Networks. In *Proceedings of the 3rd International Workshop on Algorithms and Models for the Web-Graph*, 2004.
19. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic Models for the Web Graph. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, 2000.
20. C. McDiarmid. *Concentration*. In M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed (eds.), Probabilistic Methods in Algorithmic Discrete Mathematics, Springer, 1998, pp. 195-248.
21. P. Erdős and A. Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.
22. J. Zeng and W.-J. Hsu and S. Zhou. Construction of Scale-Free Networks with Partial Information. Available at http://www.cais.ntu.edu.sg/∼zjy. 2005.

Radio Networks with Reliable Communication

Yvo Desmedt[1,⋆], Yongge Wang[2,⋆⋆], Rei Safavi-Naini[3], and Huaxiong Wang[4]

[1] University College London
y.desmedt@cs.ucl.ac.uk
[2] University of North Carolina at Charlotte
yonwang@uncc.edu
[3] University of Wollongong
rei@uow.edu.au
[4] Macquarie University
hwang@comp.mq.edu.au

Abstract. Problems of secure communication and computation have been studied extensively in network models, for example, Franklin and Yung have studied secure communications in the general networks modeled by hypergraphs. Radio networks have received special attention in recent years. For example, the Bluetooth and IEEE 802.11 networks are all based on radio network technologies. In this paper, we use directed colored-edge multigraphs to model the radio networks and study reliable and private message transmissions in radio networks.

Keywords: radio network, privacy, reliability

1 Introduction

If two parties are connected by a private and authenticated channel, then secure communication between them is guaranteed. However, in most cases, many parties are only indirectly connected, as elements of an incomplete network of private and authenticated channels. In other words they need to use intermediate or internal nodes. Achieving participants cooperation in the presence of faults is a major problem in distributed networks. The interplay of network connectivity and secure communication have been studied extensively (see, e.g., [2, 5, 8, 9, 15]). For example, Dolev [8] and Dolev et al. [9] showed that, in the case of k Byzantine faults, reliable communication is achievable only if the system's network is $2k + 1$ connected. Hadzilacos [15] has shown that connectivity $k+1$ is required to achieve reliable communication in the presence of k faulty participants even if those faults are not malicious.

Goldreich, Goldwasser, and Linial [14], Franklin and Yung [13], Franklin and Wright [12], and Wang and Desmedt [19] have initiated the study of secure communication and secure computation in *multi-recipient (multicast)* models. A "multicast channel" (such as Ethernets) enables one participant to send the same message – simultaneously and privately – to a fixed subset of participants. Franklin and Yung [13]

⋆ A part of this research was done while visiting the University of Wollongong. A part of this work has been funded by CCR-0209092. The author is BT Professor of Information Security.
⋆⋆ A part of this research was funded by NSF.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 156–166, 2005.

have given a necessary and sufficient condition for individuals to exchange private messages in multicast models in the presence of passive adversaries (passive gossipers). For the case of active Byzantine adversaries, many results have been presented by Franklin and Wright [12], and, Wang and Desmedt [19]. Note that Goldreich, Goldwasser, and Linial [14] have also studied fault-tolerant computation in the public multicast model (which can be thought of as the largest possible multirecipient channels) in the presence of active Byzantine adversaries. Specifically, Goldreich, Goldwasser, and Linial [14] has made an investigation of general fault-tolerant distributed computation in the full-information model. In the full information model no restrictions are made on the computational power of the faulty parties or the information available to them. (Namely, the faulty players may be infinitely powerful and there are no private channels connecting pairs of honest players). In particular, they present efficient two-party protocols for fault-tolerant computation of any bivariate function.

There are many examples of multicast channels. A simple example is a local area network like an Ethernet bus or a token ring. Another example is the Bluetooth or IEEE 802.11 network.

We consider a *radio network* in which stations can communicate with each other using frequencies allocated to them. Let F be the set of frequencies. Each station knows a subset of F. However at any given time it can only use a subset of its allocated frequencies, according to a defined *frequency schedule*. Communication can be *jammed* due to *intentional or accidental jamming*. The aim of this paper is to analyze these networks and construct protocols that allow reliable communication when it is possible.

The radio networks studied in [1] is similar to our model. In particular, they considered a special case of jamming as follows: a processor receives no messages if it is the recipient of of two or more partial broadcasts simultaneously. However, they do not consider privacy.

Note that a special case of frequencies allocation problem is the random key predestribution problem. Recently, Eschenauer and Gligor [11] constructed a specific random key distribution scheme and used it to build random sensor networks.

The outline of the paper is as follows. We introduce our model in Section 2. In Sections 3, 4, and 5, we study reliable message transmission against passive adversaries, jamming adversaries, and active adversaries in radio networks respectively. We study probabilistically reliable and perfectly private message transmission in certain radio networks in Section 7, and discuss the radio networks with minimal number of frequencies in Section 8. We conclude our paper with some open problems in Section 9.

2 Model

A *radio network* is a *directed colored-edge multigraph* $R(V, E, F, c)$, where V is the node set (corresponding to radio stations), E is the directed edge set (there might be more than one directed edge from one node to another one), F is the frequency (color) set, and c is a map from E to F (the map c assigns a frequency to each edge).

In a radio network, we assume that any message sent by a node v on a frequency f will be received identically by all nodes u such that there is a directed edge $e \in E$ from v to u and $c(e) = f$, whether or not v is faulty, and no other party (even if it has an incoming edge with frequency f originated from another node or it can use frequency f to broadcast to other nodes) learns anything about the content of the message.

Franklin and Yung [13] used hypergraphs[1] to model the multicast networks. A hypergraph H is a pair (V, E) where V is the node set and E is the hyperedge set. Each hyperedge $e \in E$ is a pair (v, v^*) where $v \in V$ and v^* is a subset of V. In a hypergraph, any message sent by a node v will be received identically by all nodes in v^*, whether or not v is faulty, and all parties outside of v^* learn nothing about the content of the message. Unless specified otherwise, we will use radio networks throughout our paper and will not use hypergraph networks.

It is easy to see that Franklin-Yung's hypergraph networks is a special case of our radio networks (the difference will be clear from the adversary model which we will give later).

Let $v, u \in V$ be two nodes of the radio network $R(V, E, F, c)$. We say that there is a "direct link" from node v to node u if there exists a directed edge e from v to u. We say that there is an "undirected link" from v to u if there is a directed link from v to u or a directed link from u to v. If there is a directed (undirected) link from v_i to v_{i+1} for every i, $0 \leq i < k$, then we say that there is a "directed path" ("undirected path") from v_0 to v_k.

Throughout the paper, we consider receiver-jamming, sender-jamming, destroy-jamming, and multicast as our only communication primitives.

1. A node v can receiver-jam on a frequency f if there is a directed edge e from v to some node u with $c(e) = f$. The result of receiver-jamming by v on frequency f is that for any node u such that there is a directed edge e from v to u, u cannot receive any message transmitted on the frequency f by any node.
2. A node v can sender-jam on a frequency f if there is a directed edge e from v to some node u with $c(e) = f$. The result of sender-jamming by v on frequency f is that for any node u such that there is a directed edge e from v to u, u cannot send any message on the frequency f to any node.
3. A node v can destroy-jam on a frequency f if there is a directed edge e from v to some node u with $c(e) = f$. The result of destroy-jamming by v on frequency f is that for any node u such that there is a directed edge e from v to u, u cannot receive or send any message on any frequency.
4. A message that is multicast by a node v on a frequency f in a radio network $R(V, E, F, c)$ shall be received by all nodes u satisfying the following conditions with privacy (that is, other nodes learn nothing about what was sent) and authentication (that is, the node u is guaranteed to receive the value that was multicast and to know which node multicast it)[2]
   - There is a directed edge e from v to u and $c(e) = f$.
   - u is not being jammed on the frequency f.

In addition to the intentional jamming by a malicious adversary, communications in radio networks can be accidentally jammed by honest users when a well planned schedule is not followed. Consider the following scenario, if both nodes u and v try to send messages to the node w on the same frequency f at the same time slot, then it is clear that

[1] Franklin-Yung's hypergraphs are different from the the standard definition in [3].

[2] Note that this is reasonable assumption if both u and v can share a private key. However, if u and v does not share a private key, then no authenticity is guaranteed since nodes v' might impersonate v if there is a directed edge e' from v' to u.

the node w will be "jammed". We call this kind of jamming *accidental jamming*. Accidental jamming is more or less a design problem and we will not further our study on this topic in this paper (more details could be found in [4]).

We assume that all nodes in the radio network know the complete protocol specification and the complete structure of the radio network. In a message transmission protocol, the sender A starts with a message M^A drawn from a message space $\mathcal{M}$ with respect to a certain probability distribution. At the end of the protocol, the receiver B outputs a message M^B. We consider a synchronous system in which messages are sent via multicast in rounds. During each round of the protocol, each node receives any messages that were multicast for it at the end of the previous round, flips coins and perform local computations, and then possibly multicasts a message. We will also assume that the message space $\mathcal{M}$ is a subset of a finite field $\mathbf{F}$.

Throughout this paper k denotes the number of faults under the control of the adversary. We write $|S|$ to denote the number of elements in the set S. We write $x \in_R S$ to indicate that x is chosen with respect to the uniform distribution on S.

We consider three kinds of adversaries. A passive adversary (or gossiper adversary) is an adversary who can only observe the traffic through k internal nodes. A jamming adversary is an adversary who can observe the traffics through some k internal nodes and/or jam from these k internal nodes. An active adversary (or Byzantine adversary) is an adversary with unlimited computational power who can control k internal nodes. That is, an active adversary will not only listen to the traffics through the controlled nodes, but also control the message (might be jamming noise) sent by those controlled nodes. All kinds of adversaries are assumed to know the complete protocol specification, message space, and the complete structure of the radio network. At the start of the protocol, the adversary chooses the k faulty nodes. (An alternative interpretation is that k nodes are collaborating adversaries.) The power of the adversaries is listed as follows (weakest first).

$$k\text{-passive adversary} \rightarrow k\text{-jamming adversary} \rightarrow k\text{-active adversary}$$

Throughout the paper, we assume that an active adversary can mount jamming attacks automatically. We will mainly consider three kinds of jamming in this paper: receiver-jamming, receiver-and-sender-jamming, and destroy-jamming. Thus, we will respectively have three kinds of active adversaries according to their jamming ability: rj-active adversary, rsj-active adversary, and dj-active adversary.

For any execution of the protocol, let adv be the adversary's view of the entire protocol. We write $adv(M, r)$ to denote the adversary's view when $M^A = M$ and when the sequence of coin flips used by the adversary is r.

Definition 1. *1. A message transmission protocol is* δ-reliable *if, with probability at least* $1 - \delta$*, B terminates with $M^B = M^A$. The probability is over the choices of M^A and the coin flips of all nodes.*

2. *A message transmission protocol is* reliable *if it is* 0-reliable.
3. *A message transmission protocol is* ε-private *if, for every two messages M_0, M_1 and every r, $\sum_c |\Pr[adv(M_0, r) = c] - \Pr[adv(M_1, r) = c]| \leq 2\varepsilon$. The probabilities are taken over the coin flips of the honest parties, and the sum is over all possible values of the adversary's view.*

4. *A message transmission protocol is* perfectly private *if it is* 0-private.
5. *A message transmission protocol is* (ε, δ)-secure *if it is ε-private and δ-reliable.*
6. *An (ε, δ)-secure message transmission protocol is* efficient *if its round complexity and bit complexity are polynomial in the size of the network,* $\log \frac{1}{\varepsilon}$ *(if $\varepsilon > 0$) and* $\log \frac{1}{\delta}$ *(if $\delta > 0$).*

3 Achieving Perfect Privacy and Reliability Against Passive Adversaries

Let $R(V, E, F, c)$ be a radio network, and $S \subset V$ be a node set. Then the reduced radio network $R(V \setminus S, E_{V \setminus_p S}, F, c)$ is defined by letting $E_{V \setminus_p S} = E \setminus E_S^p$, where E_S^p is the set of the following directed edges:

1. all edges originated from nodes in S.
2. all incoming edges of nodes in S.
3. all edges e from u to v such that there is an edge e' from u to some node in S and $c(e) = c(e')$.

Theorem 1. *Reliable and perfectly private message transmission from u to v in a radio network $R(V, E, F, c)$ is possible against a k-passive adversary if and only if the following conditions are satisfied:*

1. *There is a directed path from u to v in $R(V, E, F, c)$.*
2. *For any k-node set S, there is an undirected path from u to v in the reduced radio network $R(V \setminus S, E_{V \setminus_p S}, F, c)$.*

Proof. The proof is the same as that in Franklin and Yung [13] for reliable and perfectly private message transmission in hypergraphs. Q.E.D.

4 Achieving Reliability Against Jamming Adversaries

We first give a sufficient and necessary condition for achieving reliability against receiver-jammers. Let $R(V, E, F, c)$ be a radio network, and $S \subset V$ be a node set. Then the radio network $R(V \setminus S, E_{V \setminus_{rj} S}, F, c)$ is defined by letting $E_{V \setminus_{rj} S} = E \setminus E_S^{rj}$, where E_S^{rj} is the set of the following directed edges:

1. all edges originated from nodes in S.
2. all edges e from u to v such that there is an edge e' from some node in S to v and $c(e) = c(e')$.

Theorem 2. *Reliable message transmission from u to v in a radio network $R(V, E, F, c)$ against a k-receiver-jamming adversary is possible if and only if for any k-node set S, there is a directed path from u to v in the reduced radio network $R(V \setminus S, E_{V \setminus_{rj} S}, F, c)$.*

Proof. If the condition is not satisfied, then there is a k-node set S such that there is no directed path from u to v in the reduced radio network $R(V \setminus S, E_{V \setminus_{rj} S}, F, c)$. Thus

if the k-receiver-jammer controls all the nodes in S and keeps receiver-jamming on all available frequencies, all message transmissions from u to v will be blocked.

If the condition of the Theorem is satisfied, then for each k-node set S, there is a directed path p_S from u to v in the reduced radio network $R(V \setminus S, E_{V \setminus_{rj} S}, F, c)$. Thus u can transmit the message along all such paths (there are $\binom{|V|-2}{k}$ such paths) with different $\binom{|V|-2}{k}$ time slots. Q.E.D.

Now we give similar necessary and sufficient conditions for achieving reliability against receiver-and-sender-jammers and destroy-jammers. Let $R(V, E, F, c)$ be a radio network, and $S \subset V$ be a node set. Then the radio network $R(V \setminus S, E_{V \setminus_{rsj} S}, F, c)$ is defined by letting $E_{V \setminus_{rsj} S} = E \setminus E_S^{rsj}$, where E_S^{rsj} is the set of the following directed edges:

1. all edges originated from nodes in S.
2. all edges e from u to v such that there is an edge e' from some node in S to v or u and $c(e) = c(e')$.

Similarly, the radio network $R(V \setminus S, E_{V \setminus_{dj} S}, F, c)$ is defined by letting $E_{V \setminus_{dj} S} = E \setminus E_S^{dj}$, where E_S^{dj} is the set of the following directed edges:

1. all edges originated from nodes in S.
2. all edges e from u to v such that there is an edge e' from some node in S to v or u.

Theorem 3. *Reliable message transmission from u to v in a radio network $R(V, E, F, c)$ against a k-receiver-and-sender-jamming adversary (resp. k-destroy-jamming adversary) is possible if and only if for any k-node set S, there is a directed path from u to v in the reduced radio network $R(V \setminus S, E_{V \setminus_{rsj} S}, F, c)$ (resp. $R(V \setminus S, E_{V \setminus_{dj} S}, F, c)$).*

Proof. The proof is the same as that of Theorem 2. Q.E.D.

In this section, we have studied receiver-only jamming, receiver-and-sender jamming, and destroy-jamming. We will not discuss sender-jamming since a sender-jammer can easily mount receiver-jamming attacks.

5 Achieving Reliability Against Active Adversaries

Definition 2. *Let $R(V, E, F, c)$ be a radio network, and $u, v \in V$ be distinct nodes of R. u, v are* k-separable *in R, $k \geq 0$, if there is a node set $W \subset V$ with at most k nodes such that any directed path from u to v goes through at least one node in W. We say that W* separates *u, v.*

We have the following result.

Theorem 4. *The nodes u, v of a radio network R is not $2k$-separable if and only if for all k-node sets $V_1 \subset V$ there is a set S_{V_1} of directed paths from u and v such that for all k-node sets $V_2 \subset V \setminus V_1$, the following conditions hold:*

- *the paths in S_{V_1} are free of nodes in V_1,*
- *there is at least one directed path in S_{V_1} which is free of the nodes in V_2.*

Proof. First consider the case when u, v are not $2k$-seperable. We shall prove that the conditions are satisfied. For any k-node set $V_1 \subset V$, let S_{V_1} be the set of all paths from u to v which are free of nodes in V_1. Now assume that there is one k-node set $V_2 \subset V$ such that all paths in S_{V_1} go through V_2. Then $V_1 \cup V_2$ separates u and v in R. That is u and v are $2k$-separable in R which is a contradiction.

For the converse observe that the conditions on the paths S_{V_1} make it impossible to have a k-node set $V_2 \subset V$ such that $V_1 \cup V_2$ separates u and v. Indeed if there where such a set $V' = V_1 \cup V_2$ to separate u and v then there would be no path in S_{V_1} free of the of V_1 and V_2. Q.E.D.

For a radio network $R(V, E, F, c)$ and a node set $S \subset V$, the reduced radio networks $R(V \backslash S, E_{V \backslash_{rj} S}, F, c)$, $R(V \backslash S, E_{V \backslash_{rsj} S}, F, c)$, and $R(V \backslash S, E_{V \backslash_{dj} S}, F, c)$ are defined in Section 4. In the next Theorem, we give a sufficient and necessary condition for achieving reliable communication against a k-active adversary over radio networks.

Theorem 5. *A necessary and sufficient condition for reliable message transmission against a k-rj-active (resp. k-rsj-active and k-dj-active) adversary from u to v is that for any s-node set S ($s < k$), u and v are not $2(k-s)$-separable in the reduced radio network $R(V \backslash S, E_{V \backslash_{rj} S}, F, c)$ (resp. $R(V \backslash S, E_{V \backslash_{rsj} S}, F, c)$ and $R(V \backslash S, E_{V \backslash_{dj} S}, F, c)$).*

Proof. First assume that for any s-node set S ($s < k$), u and v are not $2(k-s)$-separable in the reduced radio network $R(V \setminus S, E_{V \backslash_{rj} S}, F, c)$ (respectively, $R(V \setminus S, E_{V \backslash_{rsj} S}, F, c)$ and $R(V \setminus S, E_{V \backslash_{dj} S}, F, c)$). Let $\mathcal{P}$ be the set of all directed paths from u to v. The paths in $\mathcal{P}$ will be used for transmitting messages by u to v. Let m_u be a message selected by u for transmission via these paths. Now apply Theorem 4. For any s-node set S, let V_1 be a $(k-s)$-node set and $\mathcal{P}_{V_1}$ be the set of paths in $\mathcal{P} \cap R(V \setminus S, E_{V \backslash_{rj} S}, F, c)$ (resp. $\mathcal{P} \cap R(V \setminus S, E_{V \backslash_{rsj} S}, F, c)$ and $\mathcal{P} \cap R(V \setminus S, E_{V \backslash_{dj} S}, F, c)$) which are free of nodes in V_1. Then:

- If the adversary mounts jamming attacks in the s nodes from S and send malicious message from the $k - s$ nodes in V_1, v will receive the same messages m_u via all the paths in $\mathcal{P}_{V_1}$ (since the adversary is bounded to k nodes and $\mathcal{P}_{V_1}$ is free of the nodes in V_1).
- If the adversary mounts jamming attacks in the s nodes from S and send malicious messages from some node outside V_1, there is a set V_2 which contains all these nodes. By the property of $\mathcal{P}$, there will be a directed path $P \in \mathcal{P}_{V_1}$ which is free from the nodes controlled by the adversary. In this case the messages received by v via the paths $\mathcal{P}_{C_1}$ may *not* all be the same, if the adversary is active.

Assuming that v knows the jamming nodes set S, then it follows that v can distinguish the case when the message m_u is corrupted by the adversary from the case when it is not, by testing the messages received via the paths $\mathcal{P}_{V_1}$, for the s-node set S and each $(k-s)$-node set V_1. However, an active adversary may try to control a node w in such a way that it will jam on some frequencies available to w and send malicious messages on other available frequencies to w. An active adversary could also send out malicious message no matter it has been receiver-jammed or not. Thus, the node v generally cannot learn from the received messages which set is the S. To achieve reliability, u sends to v via the paths $\mathcal{P}_{V_1}$ the message m_u labeled by (S, V_1), for each s-node set S and

each $(k-s)$-node set V_1. v checks the messages received via the paths in $\mathcal{P}_{V_1}$, for each label (S, V_1). After receiving all these messages, v recover the message according the the following rules: First, for the 0-node set $S = \emptyset$ and each k-node set V_1, v tries to recover the message from the messages received from the paths in $\mathcal{P}_{V_1}$. If v succeeds then v outputs the message. Otherwise, for each possible 1-node set S (possible jammers) and each $(k-1)$-node set V_1, v tries to recover the message from the messages received from the paths in $\mathcal{P}_{V_1}$. If v succeeds, then v outputs the message. v repeat the above steps until v finds the message. From our discussion above, v will find the correct the message with 100%-reliability.

Next assume that there exists an s-node set S ($s < k$) such that u and v can be separated by a $2(k-s)$-node set W in the reduced radio network $R(V \setminus S, E_{V \setminus_{rj} S}, F, c)$ (resp. $R(V \setminus S, E_{V \setminus_{rsj} S}, F, c)$ and $R(V \setminus S, E_{V \setminus_{dj} S}, F, c)$). Suppose that π is a message transmission protocol from u to v and let $W = W_0 \cup W_1$ be a $2(k-s)$-node separation of u and v with W_0 and W_1 each having at most $k-s$ nodes. Let m_0 be the message that u transmits. The adversary will attempt to maintain a simulation of the possible behavior of u by executing π for message $m_1 \neq m_0$. In addition to controlling the nodes in S, the strategy of the adversary is to flip a coin and then, depending on the outcome, decide which set of W_0 or W_1 to control. Let W_b be the chosen set. In each execution step of the transmission protocol, the adversary sends receiver-jamming (resp. receiver-and-sender jamming and destroy-jamming) messages on all nodes in S and causes each node in W_b to follow the protocol π as if the protocol were transmitting the message m_1. This simulation will succeeds with nonzero probability. Since v does not know whether $b = 0$ or $b = 1$, at the end of the protocol v cannot decide whether u has transmitted m_0 or m_1 if the adversary succeeds. Thus with nonzero probability, the reliability is not achieved. Q.E.D.

6 Achieving Reliability and Perfect Privacy Against Active Adversaries

Theorem 6. *Reliable and perfect private message transmission from u to v in a radio network $R(V, E, F, c)$ against a k-rj-active (resp. k-rsj-active and k-dj-active) adversary is possible if for any k-node set S, reliable message transmission against a k-rj-active (resp. k-rsj-active and k-dj-active) adversary is possible in the reduced radio network $R(V \setminus S, E_{V \setminus_p S}, F, c)$, where $E_{V \setminus_p S} = E \setminus E_S^p$ and E_S^p is the set of the following directed edges:*

1. *all edges going to nodes in S.*
2. *all edges e from u to v such that there is an edge e' from u to some node in S and $c(e) = c(e')$.*

Proof. Let $\Gamma = \{S_1, \ldots, S_t\}$ be a list of all k-node subsets of V and m^u be the message that u wants to send to v. u constructs a t-out-of-t secret sharing scheme $(s_1^u, \ldots, s_t^u)$ of m^u. For each $i \leq t$, u reliably sends s_i^u to v via the reduced radio network $R(V \setminus S_i, E_{V \setminus_p S_i}, F, c)$. For each $i \leq t$, v reliably receives s_i^v on the reduced radio network $R(V \setminus S_i, E_{V \setminus_p S_i}, F, c)$. Now assume that the adversary control all nodes in S_{i_0}, then the adversary will learn no information about $s_{s_0}^u$. Thus the above protocol is perfectly

private. It suffices to show that the above protocol is reliable. It is straightforward to show that v reliably receives all correct shares $(s_1^v, \ldots, s_t^v) = (s_1^u, \ldots, s_t^u)$. Thus the above protocol is $(0, 0)$-secure. Q.E.D.

7 Probabilistically Reliable and Perfectly Private Message Transmission in Certain Radio Networks

In this section, we briefly discuss the possibility of migrating Franklin and Wright's [12] message transmission protocol from neighbor networks to radio networks. Many radio networks have the property that each station can use all available frequencies to him/her both to receive messages and to multicast messages. We call such kind of radio networks *bi-directional radio networks*. Two nodes u and v in a bi-directional radio network $R(V, E, F, c)$ is *weakly* (n, k)*-connected* if there are n paths $p_1, \ldots, p_n$ between u and v such that for any k-node set $S \subset V$, there exists a path p_i such that there is neither edge from a node in S to a node on P_i nor edge from a node on p_i to a node in S.

Theorem 7. *If two nodes u and v in a bi-directional radio network $R(V, E, F, c)$ is weakly (n, k)-connected for some $n > k$, then there is an efficient probabilistically reliable and perfectly private message transmission between u and v.*

Proof. The proof is the same as that for the corresponding result in neighbor networks by Wang and Desmedt [19]. Q.E.D.

A similar example as in Desmedt and Wang [7] can be used to show that weak (n, k)-connectivity is not a necessary condition for achieving probabilistically reliable and perfectly private message transmissions in bi-directional radio networks. Also, similar example as in Desmedt and Wang [7] shows that there is a radio network where probabilistically reliable message transmission is possible though private message transmission is impossible.

8 Minimizing the Number of Frequencies in Certain Radio Networks

In this section, we study a specific case of radio networks initially studied in a narrow context in [6]. Let $F = \{f_1, \ldots, f_m\}$, and $\mathcal{B} = \{B_1, \ldots, B_n\}$ where $B_i \subseteq F$. Assume that there are n participants, and each participants p_i is given a set of frequency set B_i. Each participant is able to send messages with any frequency $f_j \in B_i$, and each participant who has the same frequency will receive the message. This scenario can be described by the radio network $R(V, E, F, c)$ as follows: Let $V = \{p_1, \ldots, p_n\}$, $F = \cup_i B_i$, $E = \cup_i \{(p_i, p_j)_f : f \in B_i \cap B_j, i \neq j\}$, and $c((p_i, p_j)_f) = f$.

By using Theorem 2, we derive a sufficient and necessary condition for robust frequency broadcast systems against receiver-jammers. We first introduce some notations. Let $F = \{f_1, \ldots, f_m\}$, and $\mathcal{B} = \{B_1, \ldots, B_n\}$ where $B_i \subseteq F$.

- A system $(F, \mathcal{B})$ is called a *cover free family* $CFF(m, n, k)$ [10] if for any distinct $i, i_1, \ldots, i_k \leq n$, we have $B_i \not\subseteq (B_{i_1} \cup \ldots \cup B_{i_k})$.

- A system $(F, \mathcal{B})$ is called a *key distribution pattern* [16] $KDP(m, n, k)$ if for any $i_1, \ldots, i_k \leq n$ and $i, j \leq n$ (i, j are different from $i_1, \ldots, i_k$), we have $(B_i \cap B_j) \not\subseteq (B_{i_1} \cup \ldots \cup B_{i_k})$.
- A system $(F, \mathcal{B})$ is called a *semi key distribution pattern* $SKDP(m, n, k)$ if for any $i_1, \ldots, i_k \leq n$ and $i, j \leq n$ (i, j are different from $i_1, \ldots, i_k$), at least one of the following conditions holds: there exist $s_1, \ldots, s_t$ for some $0 \leq t \leq n-2$ such that for $s_0 = i$ and $s_{t+1} = j$ we have: $(B_{s_0} \cap B_{s_1}) \not\subseteq (B_{i_1} \cup \ldots \cup B_{i_k})$, $(B_{s_1} \cap B_{s_2}) \not\subseteq (B_{i_1} \cup \ldots \cup B_{i_k})$, $(B_{s_2} \cap B_{s_3}) \not\subseteq (B_{i_1} \cup \ldots \cup B_{i_k}) \ldots$, $(B_{s_t} \cap B_{s_{t+1}}) \not\subseteq (B_{i_1} \cup \ldots \cup B_{i_k})$.

Obviously a $KDP(m, n, k)$ is a $SKDP(m, n, k)$, and a $SKDP(m, n, k)$ is a $CFF(m, n, k)$.

Theorem 8. *Let $V = \{p_1, \ldots, p_n\}$ be the participant set, $F = \{f_1, \ldots, f_m\}$ be the frequency set, and $B_i \subset F$ be the frequency set assigned to the participant p_i. Then any two participants can communicate reliably in the presence of a k-receiver-jamming adversary if and only if the system $(F, \mathcal{B})$ is a semi key distribution pattern $SKDP(m, n, k)$.*

Proof. This follows from Theorem 2 and the above definitions. Q.E.D.

For practical efficient designs, we may be interested in minimizing the number of frequencies to be used while maximizing the possible number k of jammers. For any given n and k, let

- $CFF(n, k)$ denote the minimal m such that a $CFF(m, n, k)$ exists,
- $SKDP(n, k)$ denote the minimal m such that a $SKDP(m, n, k)$ exists,
- $KDP(n, k)$ denote the minimal m such that a $KDP(m, n, k)$ exists.

From [10] and [18] we know that for any given k, there exist an integer c_1 such that $c_1 \log n \leq CFF(n, k)$, and an integer c_2 such that $KDP(n, k) \leq c_2 \log n$. That is, for a given k there exist integers c_1 and c_2 such that the following inequalities hold.

$$c_1 \log n \leq CFF(n, k) \leq SKDP(n, k) \leq KDP(n, k) \leq c_2 \log n.$$

Thus it shows that there exists an infinite family of radio networks with reliable communication against receiver-jamming adversary, requiring only $O(\log n)$ frequencies for n participants (nodes). We can even give constructions of SKDP with the asymptotically optimal number of frequencies if the network topology can be designed as desired (e.g a complete network in [6]). An interesting question is: if the network topology is fixed and given, how can we design the corresponding SKDP such that the number frequencies is as small as possible? We don't know to to do it, and it seems to be a difficult problem.

9 Conclusion and Open Problems

In this paper, we have established necessary and sufficient conditions for reliable message transmissions against jamming adversaries and active adversaries. It is easy to

show that it is **NP**-hard to check whether these conditions hold for a radio network, and most of our protocols for the sufficient condition has exponential bit-complexity in the size of the radio network. A more general and natural problem is: does there exist more efficient reliable message transmission protocols when the sufficient condition is met?

References

1. N. Alon, A. Bar-Noy, N. Linial, and D. Peleg. On the complexity of radio communication. In *Proceedings of ACM STOC 1989*, pages 274–285.
2. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In: *Proc. ACM STOC, '88*, pages 1–10, ACM Press, 1988.
3. C. Berge. *Hypergraphs: Combinatorics of finite sets*. Translated from the French. North-Holland Mathematical Library 45, 1989.
4. D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, Inc., 1992.
5. D. Chaum, C. Crepeau, and I. Damgard. Multiparty unconditional secure protocols. In: *Proc. ACM STOC '88*, pages 11–19, ACM Press, 1988.
6. Y. Desmedt, R. Safavi-Naini, H. Wang, L.M. Batten, C. Charnes and J. Pieprzyk. Broadcast anti-jamming systems. *Computer Networks*, **35**(2-3): 223-236, 2001.
7. Y. Desmedt and Y. Wang. Perfectly secure message transmission revisited. In *Proc. EuroCrypt'02*, pages 502-517. Lecture Notes in Computer Science 2332, Springer-Verlag.
8. D. Dolev. The Byzantine generals strike again. *J. of Algorithms*, **3**:14–30, 1982.
9. D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *J. of the ACM*, **40**(1):17–47, 1993.
10. P. Erdös, P. Frankl, and Z. Furedi. Families of finite sets in which no set is covered by the union of r others. *Israel Journal of Mathematics*. **51**:79-89, 1985.
11. L. Eschenauer and V. Gligor. A key-management scheme for distributed sensor networks. In: *Proc. 9th ACM Conference on Computer and Communication Security*, pages 41–47, 2002.
12. M. Franklin and R. Wright. Secure communication in minimal connectivity models. *Journal of Cryptology*, **13**(1):9–30, 2000.
13. M. Franklin and M. Yung. Secure hypergraphs: privacy from partial broadcast. In: *Proc. ACM STOC '95*, pages 36–44, ACM Press, 1995.
14. O. Goldreich, S. Goldwasser, and N. Linial. Fault-tolerant computation in the full information model. *SIAM J. Comput.* **27**(2):506–544, 1998.
15. V. Hadzilacos. *Issues of Fault Tolerance in Concurrent Computations*. PhD thesis, Harvard University, Cambridge, MA, 1984.
16. C. J. Mitchell and F. C. Piper. Key Storage in Secure Networks. *Discrete Applied Mathematics.* **21**: 215-228, 1988.
17. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In: *Proc. ACM STOC '89*, pages 73–85, ACM Press, 1989.
18. D. R. Stinson, T. van Trung and R. Wei. Secure frameproof codes, key distribution patterns, group testing algorithms and related structures. *J. Statist. Plan. Infer.* **86**:595-617, 2000.
19. Y. Wang and Y. Desmedt. Secure communication in multicast channels: the answer to Franklin and Wright's question. *J. of Cryptology*, **14**(2):121–135, 2001.

Geometric Network Design with Selfish Agents

Martin Hoefer[1,⋆] and Piotr Krysta[2,⋆⋆]

[1] Department of Computer & Information Science, Konstanz University
Box D 67, 78457 Konstanz, Germany
hoefer@inf.uni-konstanz.de
[2] Department of Computer Science, Dortmund University
Baroper Str. 301, 44221 Dortmund, Germany
piotr.krysta@cs.uni-dortmund.de

Abstract. We study a geometric version of a simple non-cooperative network creation game introduced in [2], assuming Euclidean edge costs on the plane. The price of anarchy in such geometric games with k players is $\Theta(k)$. Hence, we consider the task of minimizing players incentives to deviate from a payment scheme, purchasing the minimum cost network. In contrast to general games, in small geometric games (2 players and 2 terminals per player), a Nash equilibrium purchasing the optimum network exists. This can be translated into a $(1+\epsilon)$-approximate Nash equilibrium purchasing the optimum network under more practical assumptions, for any $\epsilon > 0$. For more players there are games with 2 terminals per player, such that any Nash equilibrium purchasing the optimum solution is at least $\left(\frac{4}{3}-\epsilon\right)$-approximate. On the algorithmic side, we show that playing small games with best-response strategies yields low-cost Nash equilibria. The distinguishing feature of our paper are new techniques to deal with the geometric setting, fundamentally different from the techniques used in [2].

1 Introduction

The Internet is a powerful and universal artefact in human history and one of the most dynamic driving forces in modern society. An interesting recent research direction is to understand and influence the development of the Internet. A fundamental difference to other networks is that the Internet is built and maintained by a number of independent agents that pursue relatively limited, selfish goals. This motivated a lot of the research in a field now called algorithmic game theory. A major direction in this field is to analyze stable solutions in non-cooperative (networking) games. The most prominent measure is the *price of anarchy* [13], which is the ratio of the worst cost of a Nash equilibrium over the cost of an optimum solution. The price of anarchy has been considered in a variety of fields, such as, load balancing [6, 13], routing [16], and flow control [8]. A slightly different measure – the cost of the best Nash equilibrium instead of the worst was considered in [17]. This is the optimum solution no user has an incentive to defect

⋆ This author is supported by the DFG Graduiertenkolleg 1042 "Explorative Analysis and Visualization of Large Information Spaces" and partially supported by DFG grant Kr 2332/1-1 within Emmy Noether program

⋆⋆ This author is supported by DFG grant Kr 2332/1-1 within Emmy Noether program.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 167–178, 2005.

from, hence we will follow [1] and refer to it as the *price of stability*. In this paper we will consider both prices for the geometric version of a network creation game.

Network Connection Games. Anshelevich et al. [2] proposed a game theoretic model called a *connection game* for building and maintaining the Internet topology, which will be the basis for our paper. Agents are to build a network, and each agent holds a number of terminals at nodes in a graph, which she wants to connect by buying edges of the graph. The cost of edges can be shared among the players. An edge can only be used for connection if fully paid for. However, once it is paid for, any player can use it to connect her terminals. A strategy of a player is a payment function, i.e., her (possibly zero) contribution to paying the cost of each edge. Given strategies of all players form a Nash equilibrium if no player could deviate to a different strategy resulting in a smaller total payment to this player. In this game the problem of finding the cheapest payment strategy for one player is the classic Steiner tree problem. The problem of finding a minimum cost network satisfying all connections and minimizing the sum of all players payments is the Steiner forest problem.
Unfortunately, both the price of anarchy and the price of stability of this game can be in the order of k, the number of players. This is also an upper bound, because if the price of anarchy were more than k, there would be a player that could deviate by purchasing the optimum network all by herself. It is NP-complete to determine that a given game has a Nash equilibrium. Thus, in [2] a different approach was taken, in which a central institution determines a network and payment schemes for players. The goal is twofold: on one hand a cheap network should be purchased, on the other hand each player shall have the least motivation to deviate. As a strict Nash equilibrium might not exist, a payment scheme was presented that determines a 3-approximate Nash equilibrium on the socially optimum network. Finding the minimum cost network, however, is NP-hard. But, approximation algorithms for the Steiner tree (forest) problem [11, 15] can be used to find a $(4.65 + \epsilon)$-approximate Nash equilibrium purchasing a 2-approximate network in polynomial time [2].
A different network creation game was proposed by Fabrikant et al. [9]. Here each player corresponds to a node, and she can only contribute to edges incident to her node. A similar game was also considered by [4, 12] in the context of social networks. Being well-suited in this setting, for the global context of the Internet it is more appropriate to assume that players hold more terminals, can share edge costs and can contribute to costs anywhere in the network.
In a more recent paper Anshelevich et al. [1] have proposed a slightly different setting for the connection game. Here the focus is put on the Shapley value, a classic cost allocation protocol. Each edge is assumed to be shared equally among the players using it, and an $O(\log k)$ upper bound on the price of stability is shown. They considered bounds on the convergence of best-response dynamics and also studied versions of the game with edge latencies and weighting schemes.

Our Contributions and Results. In this paper we consider a special case of the connection game, the *geometric* connection game. Geometric edge costs present an interesting special case of the problem, as the connection costs of a lot of large networks can be approximated by the Euclidean distance on the plane [7]. Furthermore, for the geometric versions of combinatorial optimization problems usually improved results can

be derived. For example, the geometric Steiner tree problem allows a PTAS [3], which contrasts the inapproximability for the general case [5]. This makes consideration of the geometric connection game attractive, and yields hope for significantly improved properties. In this paper, we present the following results for geometric connection games:

- The price of anarchy for geometric connection games with k players is k, even if we have two terminals per player. This, unfortunately, is the same bound as for general connection games [2].
- For games with 2 players each with 2 terminals, the price of stability is 1. The equilibrium payment scheme assigns payments along an edge according to a continuous function. For cases, in which this is unreasonable, we split an edge into small pieces, and each piece is bought completely by one player. Then a $(1+\epsilon)$-approximate Nash equilibrium can be achieved, for any $\epsilon > 0$.
- One cannot obtain results as above for more complicated games. Namely, for games with three or more players and 2 terminals per player, these results cannot be extended. There is a lower bound of $\left(\frac{4}{3}-\epsilon\right)$, for any $\epsilon > 0$, on approximate Nash equilibria purchasing the optimum network, which is slightly lower than the $\left(\frac{3}{2}-\epsilon\right)$ bound for general connection games in [2]. Thus, our result for geometric games with 2 players and 2 terminals per player is tight.
- If players play the game iteratively with best-response deviations, then in games with 2 players and 2 terminals per player the dynamics arrive at a Nash equilibrium very quickly. Furthermore, the created network is a $\sqrt{2}$-approximation to the cost of an optimum network.

The main difficulty when dealing with these geometric games is due to their inherent continuous nature. Most of our results require specific geometric arguments and new proof techniques that are fundamentally different from the ones previously used by Anshelevich et al. [2] for general connection games. The development of these new techniques is considered as an additional contribution of our paper.

Outline. Section 2 contains a formal definition of the geometric connection game, and Section 3 presents our results on the price of anarchy. Section 4 describes the results on the price of stability (Theorems 2, 3, and 5), and the analysis of the best-response dynamics (Theorem 4). Missing proofs will be given in the full version of the paper.

2 The Model and Preliminaries

The geometric connection game is defined as follows. Let V be a set of nodes which are points in the Euclidean plane. There are k non-cooperative players, each holding a number of terminals located at a subset of nodes from V. Each player strives to connect all of her terminals into a connected component. To achieve this a player offers money to purchase segments in the plane. The cost of a segment equals its length in the plane. Once the total amount of money offered by all players for a certain segment exceeds its cost, the segment is considered *bought*. Bought segments can be used by *all* players to connect their terminals, even if they contribute nothing to their costs. A strategy of player i is a payment function p_i that specifies how much she contributes to each segment in the plane. A collection of strategies, one for each player, is called

a *payment scheme* $p = (p_1, \ldots, p_k)$. A Nash equilibrium is a payment scheme p, in which no player i can connect her terminals at a lower cost by unilaterally reallocating her payments and switching to another function p_i' [1]. We will denote the social optimum solution, i.e., the minimum cost forest that connects the terminals of each player, by T^*. The subtree of T^* needed by player i to connect her terminals is denoted by T^i.

Constructing a minimum cost network satisfying all connections is the geometric Steiner forest problem. As the components of a Steiner forest are Steiner trees for a subset of players, well-known properties of optimum geometric Steiner trees hold for T^*.

Lemma 1. *[10, 14] Any 2 adjacent edges in an optimal geometric Steiner tree connect with an inner angle of at least 120°.*

Hence, every Steiner point of an optimal geometric Steiner tree has degree 3 and each of the 3 edges meeting at it makes angles of 120° with the other two [10, 14].
Another powerful tool for the analysis of connection games is the notion of a *connection set* that was the key ingredient to the analysis presented in [2].

Definition 1. *A connection set S of player i is a subset of edges of T^i, such that for each connected component C in $T^* \setminus S$ either (1°) there is a terminal of i in C, or (2°) any player that has a terminal in C has all of its terminals in C.*

Intuitively, after removing a connection set from T^* and reconnecting the terminals of player i the terminals of all players will be connected in the resulting solution. As T^* is the optimal solution, the maximum cost of *any* connection set S for player i is a lower bound for the cost of *any* of her deviations. Connection sets in a game with 2 terminals per player are easy to determine. Each T^i forms a path inside T^*, and two edges e, e' belong to the same connection set for player i iff $i \in \{j \in \{1, \ldots, k\} : e \in T^j\} = \{j' \in \{1, \ldots, k\} : e' \in T^{j'}\}$. We will mainly deploy geometric arguments and use connection sets only to limit the number of cases to be examined.

3 The Price of Anarchy

Theorem 1. *The price of anarchy for the geometric connection game with k players and 2 terminals per player, is precisely k.*

Proof. We have already argued in the introduction that the price of anarchy is at most k. Let us show now that k is also a lower bound. At first we will somewhat generally consider how a player in the geometric environment is motivated to deviate from a given payment scheme. Suppose we are given a game with 2 terminals per player and a feasible forest T, which satisfies the connection requirement for each player. Furthermore, let p be a payment function, which specifies a payment for each player on each edge. The next two Lemmas 2 and 3 follow directly from the Triangle Inequality.

Lemma 2. *If the deviation for a player i from p includes an edge $e \notin T$, this edge is a straight line segment, with start and end either at a terminal or some other part of T (possibly an interior point of some edge of T). It is located completely inside the Euclidean convex hull of T.*

[1] An α-approximate Nash equilibrium is a payment scheme where each player may reduce her costs by at most a factor of α by deviating.

Using these observations we can specify some properties of Nash equilibria for geometric connection games.

Lemma 3. *In a Nash equilibrium of the geometric connection game for k players, edges e_1, e_2 bought fully by one player are straight segments and meet with other, differently purchased edges with an inner angle of at least 90°. In the case of 2 terminals per player e_1 and e_2 can only meet at a point if they have an inner angle of* 180°.

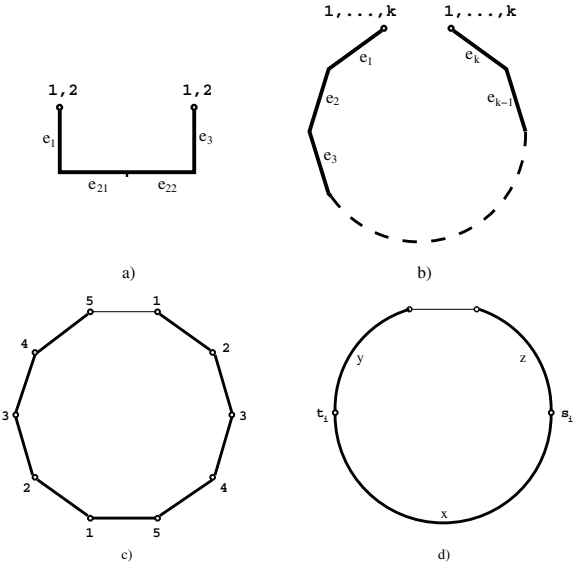


Fig. 1. (a),(b) Geometric games with maximum price of anarchy; (c),(d) Lower bound for approximate Nash purchasing T^*

Consider the game for 2 players and 2 terminals shown in Figure 1a. We have two designated nodes, each containing one terminal of each player. Let $e_2 = e_{21} \cup e_{22}$. The payment scheme purchases T in the following way. Player 1 pays for e_3 and e_{21}. Player 2 pays for e_1 and e_{22}. Let the costs be $e_1 = e_3 = e_{21} = e_{22} = \frac{1}{2}$. e_1 and e_2 as well as e_2 and e_3 are orthogonal. The optimal solution in this network is the direct connection between the terminals. The presented payment scheme, however, forms a Nash equilibrium. Note that the necessary conditions of Lemma 3 are fulfilled. In addition, no player can deviate by simply removing any payment from the network. Lemma 2 restricts the attention to straight segments inside the rectangle, which is the Euclidean hull of T. The argument is given for player 1 – it can be applied symmetrically to player 2. We will consider all meaningful straight segments inside the convex hull of T as deviations. Note that any deviation with both endpoints inside the same edge e_1, e_{21}, etc. (or with endpoints in e_{21} and e_{22}) and any segment between e_1 and e_3 is unprofitable. Now consider a deviation $d = (u, v)$ for player 1 connecting points $u \in e_1$ and $v \in e_{22}$, which are the two segments paid for by player 2. Suppose $d \neq e_{21}$ then $|d| > \frac{1}{2}$. Using d, however, player 1 can save only a cost of $\frac{1}{2}$ by dropping e_{21}.

If $u \in e_3$ and $v \in e_{21}$, then d connects segments purchased by player 1. Suppose she defects to such an edge. Let e_3^d be the part of e_3 inside the cycle introduced by d in T (e_{21}^d accordingly). Then with the Phythagorean Theorem and $|e_3^d|, |e_{21}^d| \leq \frac{1}{2}$ the lower bound $|d| \geq |e_{21}^d| + \frac{1}{2} \geq |e_3^d| + |e_{21}^d|$ holds, so d is not profitable for player 1. Hence, all edges player 1 would consider for a deviation are unprofitable. With the symmetric argument for player 2 it follows that the payment scheme represents a Nash equilibrium. Since the optimum solution is half of the cost of T, the theorem follows for games with 2 players and 2 terminals per player.
In a network with more players assume that each player has one terminal at each of the two designated nodes. The nodes are separated by a distance of 1. Construct a path between the nodes, which approximates a cycle with k straight edges of cost 1 each (see Figure 1b). Each player i is assigned to pay for one edge e_i of cost 1. Observe that the necessary conditions of Lemma 3 are fulfilled. Now consider the deviations for a player i. She will neither consider segments that cost more than 1 nor segments that do not allow her to save on e_i. Of the remaining deviations none will yield any profit, because the cyclic structure makes the interior angles between the edges amount to at least $90\,^\circ$. Any deviation $d = (u, v)$ from a point $u \in e_i$ to any other point v will be longer than the corresponding part e_i^d that it allows to save. As the optimum solution is the direct connection of cost 1, Theorem 1 follows. □

This result is contrasted with a result on the price of stability, i.e., the cost of the best Nash equilibrium over the cost of the optimum network.

4 The Price of Stability

Theorem 2. *The price of stability for geometric connection games with 2 players and 2 terminals per player is 1.*

Proof. We will consider all different games classifying them by the structure of their optimum solution network. The networks in Figure 2 depict the different structures of T^* we consider. In each of them there is an edge $e_3 \in T^1$ and $e_3 \in T^2$. If there is no such edge, the solution is composed of one connection set per player, and a Nash equilibrium can be derived by assigning each player to purchase her subtree T^i.

Type 21 In this case the network is a path (see Figure 2a). The following Lemma 4 describes the structure of meaningful deviations. Nash equilibrium requires that everybody contributes only inside her subtree. That means e_1 is paid by player 1 and e_2 by player 2. The length of every segment lower bounds the deviation costs (by the connection set property)–so no deviation between points of the same segment is meaningful. Furthermore, from the above we know that only straight segments inside the convex hull need to be considered. Hence, the only cases left are described in the lemma below.

Lemma 4. *Given an optimal network T of Type 21 and a payment function p that assigns player i only to pay for edges in T^i, the only deviations for player 1 are straight segments from a point $u \in e_1$ to a point $v \in e_3$, player 2 only from $u \in e_2$ to $v \in e_3$.*

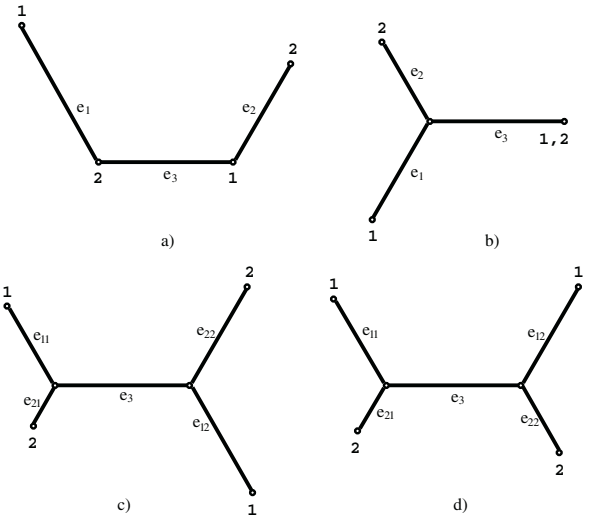


Fig. 2. Network types for T^*

Proof. (sketch) We analyze the payments of player 1. The claim follows for player 2 by symmetry. By Lemma 2 and the fact that the segments of T^* are straight we can restrict the possible deviations to 3 possible cases of straight segments $d = (u, v)$: (1) $u \in e_1$, $v \in e_3$; (2) $u \in e_1$, $v \in e_2$; (3) $u \in e_2$, $v \in e_3$. Cases 2 and 3 can both be disregarded, because either they allow a player to deviate from (parts of) only one connection set, or they can be decomposed into or bounded by deviations from Case 1. □

An adjusted version of Lemma 4 will be true for most of the cases we consider in the remaining proof (cf. Lemmas 5, 6 and 7). Proofs of these lemmas will be omitted.
By the Cosine Theorem the deviation lengths between two adjacent segments are minimized if the angle between segments is minimized, i.e., amounts to 120 ° (cf. Lemma 1). Hence, for the remaining proof we will use an
Angle Assumption: all the edges connecting in the optimal solution make inner angles of exactly 120 °.
The following payment scheme forms a Nash equilibrium. Let $e_{3,1}$ be a half subsegment of e_3 connecting the center of e_3 with the terminal of player 1. Similarly, $e_{3,2}$ is the other half subsegment of e_3, connecting the center of e_3 with the terminal of player 2. Then, for player 1, $p_1(e_1) = |e_1|$, $p_1(e_{3,1}) = |e_{3,1}|$, and $p_1 = 0$ elsewhere. For player 2, $p_2(e_2) = |e_2|$, $p_2(e_{3,2}) = |e_{3,2}|$, and $p_2 = 0$ elsewhere.
Note first that the necessary conditions from Lemma 3 are fulfilled. Consider a deviation $d = (u, v)$ in Lemma 4 for player 1 with $u \in e_1$ and $v \in e_3$. As the angle between e_1 and e_3 is exactly 120 °, the length (and cost) of this segment by the Cosine Theorem is

$$|d| = \sqrt{|e_1^d|^2 + |e_3^d|^2 + |e_1^d||e_3^d|},$$

where e_1^d and e_3^d are the segments of e_1 and e_3 in the cycle in $T + d$. The payment of player 1 that can be removed when buying d is $p_1(e_1^d) + p_1(e_3^d) = |e_1^d| + \max(|e_3^d| - \frac{|e_3|}{2}, 0)$. Once v lies in $e_{3,2}$ paid by player 2, $|e_3^d| < \frac{|e_3|}{2}$ and the deviation cannot be cheaper than $|e_1^d|$. Otherwise when $|e_3^d| \geq \frac{|e_3|}{2}$ we can see that $|e_1^d||e_3| + |e_3^d||e_3| - |e_1^d||e_3^d| \geq \frac{|e_3|^2}{4}$. Then it follows that

$$|e_1^d|^2 + |e_3^d|^2 + |e_1^d||e_3^d| \geq |e_1^d|^2 + |e_3^d|^2 + \frac{|e_3|^2}{4} - |e_1^d||e_3| - |e_3^d||e_3| + 2|e_1^d||e_3^d|.$$

Finally we get $|d| \geq |e_1^d| + |e_3^d| - \frac{|e_3|}{2} = p_1(e_1^d) + p_1(e_3^d)$ and see that player 1 has no way of improving her payments. By symmetry the same is true for player 2 and the proof for this network type is completed. □

Type 22 This network type consists of a star, which has a Steiner vertex in the middle and three leaves containing the terminals of the players (see Figure 2b).

Lemma 5. *Given an optimal network T of Type 22 and a payment function p that assigns player i only to pay for edges in T^i, the only deviations for player 1 are straight segments from a point $u \in e_1$ to a point $v \in e_3$, player 2 only from $u \in e_2$ to $v \in e_3$.*

The following is a Nash equilibrium payment scheme. Let $e_3' = (u, v)$, with $u, v \in e_3$, be any subsegment of e_3, where u, v are two interior points on e_3. Then, in the strategy for player 1, $p_1(e_1) = |e_1|$ and $p_1(e_3') = \frac{|e_3'|}{2}$ for any such subsegment e_3' of e_3. For player 2, $p_2(e_2) = |e_2|$ and $p_2(e_3') = p_1(e_3')$ for any subsegment e_3' of e_3. For any other segments in the plane, $p_1 = 0$ and $p_2 = 0$. Consider a deviation $d = (u, v)$ for player 1 with $u \in e_1$ and $v \in e_3$. The amount of payment player 1 can save with this edge is

$$|e_1^d| + \frac{|e_3^d|}{2} = \sqrt{|e_1^d|^2 + |e_1^d||e_3^d| + \frac{|e_3^d|^2}{4}} = \sqrt{|d|^2 - \frac{3|e_3^d|^2}{4}} < |d|.$$

Hence, the deviation is more costly than the possible cost saving for player 1. The proof of a strict Nash for this type follows from the symmetric argument for player 2. □

Type 23 In this network type we have two Steiner points and the terminals of a player are located on different sides of the line through e_3 (see Figure 2c). The connection set for player i formed by edges that are only in T^i now consist of two edges e_{i1} and e_{i2}.

Lemma 6. *Suppose we have Type 23 optimum Steiner network. Under the same assumptions as of Lemma 4 the only deviations player 1 will consider in this game are straight edges from a point $u \in e_{11}$ or a point $u \in e_{12}$ to a point $v \in e_3$, player 2 only from $u \in e_{21}$ or $u \in e_{22}$ to $v \in e_3$.*

We construct an equilibrium payment as follows. For player 1, $p_1(e_{11}) = |e_{11}|$, $p_1(e_{12}) = |e_{12}|$ and $p_1(e_3') = \frac{|e_3'|}{2}$ with e_3' being any subsegment of e_3, as for the scheme of Type 22 above. For player 2, $p_2(e_{21}) = |e_{21}|$, $p_2(e_{22}) = |e_{22}|$ and $p_2(e_3') = p_1(e_3')$. Otherwise, $p_1 = 0$ and $p_2 = 0$. For all possible deviations, the cost is greater than the contribution to T^* a player could save. This follows with the proof of Type 22. □

Type 24 The last network type considered is the one including two Steiner points where the terminals of a player are located on the same side of the line through e_3 (see Figure 2d). Here we get some additional deviations that complicate the analysis.

Lemma 7. *Given a network T of Type 24 and a payment function that assigns payments to player i only in her subtree T^i. Then the only deviations player 1 considers are straight edges between $u \in e_{11}$ or $u \in e_{12}$ and $v \in e_3$ as well as the direct connection between her terminals. For player 2 the symmetric claim holds.*

To present the payment function, we scale our game such that e_3 has length 1. We now treat e_3 as an interval $[0, 1]$ and introduce a function $f(x, y) \in [0, 1], 0 \leq x \leq y \leq 1$ that specifies the fraction of the cost player 1 pays in the interval $[x, y]$ of e_3, i.e., the payment of player 1 on $[x, y]$ is $(y - x)f(x, y)$. Let, w.l.o.g., the Steiner point of e_{11} be point 0 of e_3 and the other Steiner point be point 1. We now have to ensure that for every deviation from e_{11} or e_{12} to a point $y, 1 - y \in e_3$ the savings on the segments do not exceed the cost of the deviation. This results in the bounds $|e_{11}| + yf(0, y) \leq \sqrt{|e_{11}|^2 + y^2 + |e_{11}|y}$ and $|e_{12}| + yf(1 - y, 1) \leq \sqrt{|e_{12}|^2 + y^2 + |e_{12}|y}$. For player 2 the symmetric requirements lead to similar bounds with $|e_{21}|$ and $|e_{22}|$. Furthermore, we can derive bounds from the direct connections between the terminals. They will be denoted as d_1 and d_2 for players 1 and 2, respectively. With the optimality of our network and $|e_3| = 1$ we have

$$|d_1| + |d_2| \geq |e_{11}| + |e_{12}| + |e_{21}| + |e_{22}| + 1. \tag{1}$$

As we strive for a Nash payment scheme, d_1 and d_2 are not cheaper than the contribution of the players, hence $|d_1| \geq |e_{11}| + |e_{12}| + f(0, 1)$ and $|d_2| \geq |e_{21}| + |e_{22}| + 1 - f(0, 1)$. The nature of these edges implies that their bounds only apply to the payment on the whole segment e_3, i.e., they do not restrict the partition of the payment inside the segment. Using a function $h(x) = \sqrt{x^2 + x + 1} - x$ and solving for f the previous bounds can be turned into

$$|e_{21}| + |e_{22}| + 1 - |d_2| \leq f(0, 1) \leq |d_1| - |e_{11}| - |e_{12}|, \tag{2}$$

$$1 - h\left(\frac{|e_{21}|}{y}\right) \leq f(0, y) \leq h\left(\frac{|e_{11}|}{y}\right), \tag{3}$$

$$1 - h\left(\frac{|e_{22}|}{y}\right) \leq f(1 - y, 1) \leq h\left(\frac{|e_{12}|}{y}\right). \tag{4}$$

Now consider the behavior of $h(x)$ in (3) and (4) when altering the constants $|e_{11}|$ and $|e_{12}|$. We observe that for the derivative $h'(x) < 0$ holds. The function is monotone decreasing in x, and increasing $|e_{11}|, |e_{12}|, |e_{21}|, |e_{22}|$ tightens lower and upper bounds. So we will only consider deviations from terminals to e_3, as this results in the strongest bounds for the Nash payments.

We also require that payments can be feasibly split to subintervals. The payment of player 1 on an interval $[x, y]$ has to be the sum of the payments on the two subintervals $[x, v]$ and $[v, y]$ for any $v \in [x, y]$. Using this property, we can define $f(x, y)$ by using the functions $f(0, y)$ and $f(1 - y, 1)$. Namely, for $0 \leq x \leq y \leq 1$, we have:

$$f(x,y) = \frac{yf(0,y) - xf(0,x)}{y-x}, \quad f(1-y,1-x) = \frac{yf(1-y,1) - xf(1-x,1)}{y-x}. \tag{5}$$

In particular, we will focus on symmetric payment functions, i.e., we will assume that $f(0,y) = f(1-y,1)$ for any $y \in [0,1]$. This also implies that $f(x,y) = f(1-y,1-x)$, where $0 \le x \le y \le 1$. For the rest of the proof we will provide a feasible function $f(0,y)$, which obviously must obey all bounds (2)-(4). First, we pay some attention to the feasibility of the bounds.

Lemma 8. *The bounds (2)–(4) do not imply a contradiction. In particular the interior bounds (3), (4) can be fulfilled by* $f(0,y) = \frac{1}{2}$.

Proof. We already know that the upper bound function $h(x)$ is monotone decreasing in x. We observe that for any $x, x' > 0$

$$\lim_{x\to\infty}(1-h(x)) \;=\; \frac{1}{2} \;=\; \lim_{x\to\infty} h(x), \quad \text{and} \quad 1-h(x) \;\le\; \frac{1}{2} \;\le\; h(x').$$

This proves the second part of the lemma. Regarding the first part, a negation of the bounds leads to a contradiction with the Triangle Inequality with d_1 and d_2. □

Lemma 8 supports our proofs for the previous network Types 23 and 22. The function used is the linear function $f(x,y) = \frac{1}{2}$ and satisfies bounds (3), (4), which are the only ones present. For Type 24 network a solution is possible as well. In the easiest case if

$$|e_{21}| + |e_{22}| + 1 - |d_2| \le \frac{1}{2} \le |d_1| - |e_{11}| - |e_{12}| \tag{6}$$

holds, $f(0,y) = f(x,y) = \frac{1}{2}$ again gives a Nash equilibrium payment function. Hence, for the remainder of the proof we will assume (6) is not valid. A solution for this more complicated situation is presented in the next lemma.

Lemma 9. *There is a constant* t *and one of the two functions*

$$f_1(0,y) = h\,(t/y) \quad \textit{or} \quad f_2(0,y) = 1 - h\,(t/y)$$

that allow us to construct a payment scheme forming a Nash equilibrium in a network of Type 24.

Proof. In the first case we assume that $r = |e_{21}| + |e_{22}| + 1 - |d_2| > \frac{1}{2}$. Then $f(0,y)$ must behave like the upper bounds and achieve a value of r for $y = 1$, so $f_1(0,y) = h(t/y)$ with $t = \frac{1-r^2}{2r-1}$.

In the second case we assume $r = |d_1| - |e_{11}| - |e_{12}| < \frac{1}{2}$. Then $f(0,y)$ must behave like the lower bounds and achieves a value of r for $y = 1$, so $f_2(0,y) = 1 - h(t/y)$ with $t = \frac{1-r^2}{2r-1} - 1$. To achieve a consistent definition of $f(x,y)$ we define $f(1,1) = f(0,0) := \lim_{y\to 0} f(0,y) = \frac{1}{2}$.

Then, with Lemma 8 and the monotonicity of $h(x)$ we see that the functions f_1, f_2 obey bounds (2)–(4) for any $y \in [0,1]$. Lemma 8 says that the upper bound functions of (3) and (4) map only to $[0.5, 1]$, and the lower bound functions map only to $[0, 0.5]$.

Since f_1 maps only to $[0.5, 1]$, all lower bounds are feasible. The upper bounds for f_1 are feasible by the monotonicity of $h(\cdot)$ – the constants t in f_1 are smaller than appropriate constants in the upper bounds. Similarly for f_2. Functions f_1 and f_2 allow to construct a Nash equilibrium payment function. If the payment of player 1 is given by f_1 (f_2), then the payment of player 2 is given by a function f_2 (f_1) with the same constant r as for player 1. This concludes the proofs of Lemma 9 and Theorem 2. □

The proof of the theorem requires that an edge e_3 is purchased such that the payments of players on the intervals follow a continuous differentiable function. This is a quite strong and very unrealistic property. We present two possible alternatives to avoid this. First a discretization of the payment scheme on e_3 is considered such that subsegments of the network are assigned to be purchased completely by single players. It slightly increases the incentives to deviate. Second we let players play the game according to best-response strategies. This will lead into a low-cost strict Nash equilibrium. Full proofs are omitted due to space constraints. A *divisible* payment scheme $p = (p_1, \ldots, p_k)$ for a geometric connection game is a payment scheme such that there is a partition $\mathcal{P}$ of the plane into segments with $p_i(e) = 0$ or $p_i(e) = |e|$ for all $i = 1, \ldots, k$ and all $e \in \mathcal{P}$.

Theorem 3. *Given any $\epsilon > 0$ and any geometric connection game with 2 players and 2 terminals per player, there exists a divisible payment scheme, which is a $(1 + \epsilon)$-approximate Nash equilibrium as cheap as the optimum solution.*

Theorem 4. *In any geometric connection game with 2 players and 2 terminals per player, there exists a Nash equilibrium generated by at most 3 steps of the best-response dynamics, which is a $\sqrt{2}$-approximation to the optimum solution.*

The results cannot be generalized to games with more players. For games with 3 or more players there is a constant lower bound on approximate Nash equilibria.

Theorem 5. *For any $k \geq 3, \epsilon > 0$ there exists a game with k players and 2 terminals per player, for which every optimum solution is at least a $(\frac{4k-2}{3k-1} - \epsilon)$-approximate Nash equilibrium.*

Proof. In the class of games delivering the bound there is a circle of terminals with unit distance, and the optimal solution is a minimum spanning tree of cost $2k - 1$. In the geometric environment edges crossing the interior of the circle are not of interest, because their cost is always larger than 1. So no player will consider them as a reasonable alternative. Consider the game in Figure 1c, in which every edge of T^* has cost 1. T^* is depicted with an additional edge of cost 1, which will be the only deviation edge considered. The situation for a player i can be simplified to the view of Figure 1d. Note that for players 1 and k, $z = 0$ and $y = 0$, respectively. For every player there are at least two ways to deviate, either she just contributes to one half of the cycle by paying part x of this half, or she completes the other half of the cycle by paying $y + z + 1$, where y, z are the parts she pays on the depicted portions of the cycle. Thus her deviation factor will be at least $\max\left\{\frac{x+y+z}{x}, \frac{x+y+z}{y+z+1}\right\}$. Minimizing this expression with $x = y + z + 1$ there is at least one player, who pays for $x + y + z = 2x - 1 \geq \frac{2k-1}{k}$. Solving for x and combining with $x = y + z + 1$ results in: $\frac{x+y+z}{x} = \frac{2x-1}{x} \geq \frac{4k-2}{3k-1}$. Now move the terminals s_1 and t_k a little further to the outside keeping the lengths of the edges (s_1, s_2)

and (t_{k-1}, t_k) to 1, but increasing the length of (s_1, t_k) to length $(1+\epsilon)$. T^* will then be the unique optimal solution, and the factor becomes at least $\left(\frac{4k-2}{3k-1} - \epsilon\right)$. □

References

1. E. Anshelevich, A. Dasgupta, J. Kleinberg, É. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 295–304, 2004.
2. E. Anshelevich, A. Dasgupta, É. Tardos, and T. Wexler. Near-optimal network design with selfish agents. In *Proceedings of the 35th Annual Symposium on Theory of Computing (STOC)*, pages 511–520, 2003.
3. S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.
4. V. Bala and S. Goyal. A non-cooperative model of network formation. *Econometrica*, 68:1181–1229, 2000.
5. M. Chlebík and J. Chlebíková. Approximation hardness of the Steiner tree problem in graphs. In *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 170–179, 2002.
6. A. Czumaj, P. Krysta, and B. Vöcking. Selfish traffic allocation for server farms. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 287–296, 2002.
7. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry - Algorithms and Applications*. Springer Verlag, 1997.
8. D. Dutta, A. Goel, and J. Heidemann. Oblivious AQM and Nash equilibrium. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.
9. A. Fabrikant, A. Luthera, E. Maneva, C. Papadimitriou, and S. Shenker. On a network creation game. In *Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 347–351, 2003.
10. E. Gilbert and H. Pollak. Steiner Minimal Trees. *SIAM Journal on Applied Mathematics*, 16:1–29, 1968.
11. M. Goemams and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.
12. H. Heller and S. Sarangi. Nash networks with heterogeneous agents. Technical Report Working Paper Series, E-2001-1, Virginia Tech, 2001.
13. E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 404–413, 1999.
14. Z. Melzak. On the problem of Steiner. *Canadian Mathematical Bulletin*, 4:143–148, 1961.
15. G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 770–779, 2000.
16. T. Roughgarden and É.Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002.
17. A. Schulz and N. Stier Moses. Selfish routing in capacitated networks. *Mathematics of Operations Research*, 29(4):961–976, 2004.

Bicriteria Network Design via Iterative Rounding

Piotr Krysta*

Department of Computer Science, Dortmund University
Baroper Str. 301, 44221 Dortmund, Germany
piotr.krysta@cs.uni-dortmund.de

Abstract. We study the edge-connectivity survivable network design problem with an additional linear budget constraint. We give a strongly polynomial time $(3,3)$-approximation algorithm for this problem, by extending a linear programming based technique of iterative rounding. Previously, a $(4,4)$-approximation algorithm for this problem was known. The running time of this previous algorithm is not strongly polynomial.

1 Introduction

Let $G = (V, E)$ be an undirected multi-graph with cost $c_e \geq 0$ on each edge $e \in E$. For $S \subset V$ and $E' \subseteq E$, we denote by $\delta_{G'}(S)$ the set of edges with one end in S and the other end in $V \setminus S$ in the graph $G' = (V, E')$. We omit the subscript G' in $\delta_{G'}(S)$, if $E' = E$ or if E' is clear from the context.

Edge-Connectivity Survivable Network Design Problem (EC-SNDP) is, given integer requirements r_{uv} for $u, v \in V$, $u \neq v$, to find a minimum-cost subgraph, with respect to costs c_e, containing for any u, v at least r_{uv} edge-disjoint u-v paths. EC-SNDP can be formulated as the following integer linear program (ILP), which we will call (IP1):

$$\begin{aligned} \min \quad & \textstyle\sum_{e \in E} c_e x_e & & \\ s.t. \; & \textstyle\sum_{e \in \delta_G(S)} x_e \geq f(S) & \forall S \subseteq V & \quad (1) \\ & x_e \in \{0,1\} & \forall e \in E, & \quad (2) \end{aligned}$$

where $f(S) = \max\{r_{uv} : u \in S, v \notin S\}$.

The defined problem generalizes many known graph optimization problems, e.g., the Steiner tree problem [10], Steiner forest problem [1, 3], minimum-cost k-edge-connected spanning subgraph problem [5], and others. In a breakthrough paper, Jain [4] has designed a 2-approximation algorithm for EC-SNDP by employing a technique of iterative rounding.

* The author is supported by DFG grant Kr 2332/1-1 within Emmy Noether program. This work has been partially done while visiting the Department of Computer Science of RUTGERS–The State University of New Jersey, Camden, NJ, U.S.A., May 2004.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 179–187, 2005.

When designing networks it is often natural to seek for a cheap network as in EC-SNDP that simultaneously optimizes some additional linear cost objective– a so-called budget constrained problem. Such bicriteria optimization problems have been studied by many researchers, see, e.g., [2, 6, 8, 9].

We now define a bicriteria version of EC-SNDP. Given an additional cost (length) $l_e \geq 0$ on each $e \in E$, and a number $L > 0$, we consider EC-SNDP with an additional budget constraint, $\sum_{e \in E} l_e x_e \leq L$, called Constr EC-SNDP.

Let now $opt(G)$ denote the cost of an optimum solution to a given instance G of Constr EC-SNDP, e.g., the solution with minimum cost with respect to c among all solutions with length at most L. An (α, β)-approximation algorithm for Constr EC-SNDP is a polynomial time algorithm that always finds a solution G' to the instance G of this problem, such that G' has cost at most $\alpha \cdot opt(G)$ with respect to c, and such that G' has length at most $\beta \cdot L$.

Results of Marathe et al. [6] and the 2-approximation algorithm for EC-SNDP by Jain [4] imply a $(4, 4)$-approximation algorithm for Constr EC-SNDP. This approach uses a parametric binary search and the 2-approximation algorithm as a subroutine. Thus, the running time of the resulting $(4, 4)$-approximation algorithm is not strongly polynomial. We extend the technique of iterative rounding and obtain an improved $(3, 3)$-approximation algorithm for Constr EC-SNDP. Our algorithm additionally has strongly polynomial running time.

2 A Polyhedral Theorem

This section contains the main technical ingredient of our method. To prove it, i.e., to prove Theorem 2, we need some new notions and notations – we will mostly rely on Jain's paper [4] in this respect. Nevertheless, we will define below the most important notions from [4]. The proof below also requires some basic knowledge in linear programming and polyhedral combinatorics, see, e.g., [11]. A set function $f : 2^V \rightarrow \mathcal{Z}$ is *weakly supermodular* if:

1. $f(V) = 0$, and
2. For every $A, B \subseteq V$ at least one of the following holds:
   (a) $f(A) + f(B) \leq f(A \setminus B) + f(B \setminus A)$,
   (b) $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$.

We note, that the specific set function $f(S) = \max\{r_{uv} : u \in S, v \notin S\}$ defined before for EC-SNDP is weakly supermodular, see [4]. Let $\Pi = \Pi(E)$ correspond to constraints (1)–(2), and Π' be the linear programming relaxation of Π, that is, Π' is Π with constraints (2) replaced by $x_e \in [0, 1]$ $\forall e \in E$. Thus, Π' is the EC-SNDP polytope. Given an $x \in \Pi'$, i.e., a feasible fractional solution to the EC-SNDP relaxation Π' and a set $S \subseteq V$, we define $x(\delta(S)) = \sum_{e \in \delta(S)} x_e$. For a subset $S \subseteq V$, let $\boldsymbol{x}(S) \in \{0, 1\}^{|E|}$ be the 0-1 vector of the coefficients in the left-hand-side in constraint (1). Having fixed an $x \in \Pi'$ and $S \subseteq V$, we say that set S (or the corresponding constraint (1)) is *tight* if $x(\delta(S)) = f(S)$. Given a family of sets $\mathcal{T} \subseteq 2^V$, let $sp(\mathcal{T})$ denote the vector space spanned by the vectors $\{\boldsymbol{x}(S) : S \in \mathcal{T}\}$. Finally, we say that two sets $A, B \subseteq V$ *cross* if none of the sets

$A \setminus B$, $B \setminus A$ and $A \cap B$ is empty. A family of sets is *laminar* if no two sets in it cross. Each given edge $e = (u, v) \in E$ has two *endpoints*, denoted by e_u and e_v. Since we have $|E|$ edges, we have exactly $2 \cdot |E|$ endpoints.

2.1 A First Polyhedral Theorem

We first show that even without going into the combinatorial structure of the Constr EC-SNDP polytope we are able to easily obtain a $(4, 4)$-approximation algorithm with strongly polynomial running time. Namely, we shall prove the following fact.

Theorem 1. *Let x^* be any extreme point of the EC-SNDP polytope defined by a weakly supermodular function f with an additional linear constraint lin. Then there is an index $e \in E$ such that $x^*_e \geq 1/4$.*

Proof. Let x^* be a given extreme point of the polytope as in the theorem. We will use the following well known characterization of extreme points (see, e.g., page 104 in the book by Schrijver [11]).

Lemma 1. *Let $P = \{x \in \Re^m : Ax = a, Dx \leq d\}$ be a polytope defined by a system $Ax = a, Dx \leq d$ of linear equations and inequalities. Then $\bar{x} \in P$ is an extreme point of P if and only if there exists a set of m equations and tight inequalities from the system defining P, such that $\bar{x}$ is the unique solution to the corresponding equation system.*

Thus, x^* is determined by $|E| = m$ linearly independent equations, say $\mathcal{S}^*$, from the system $\Pi' \cup \{lin\}$. (Let us recall, that Π' corresponds to constraints (1)–(2), with constraints (2) replaced by $x_e \in [0, 1] \ \forall e \in E$.)

Assume first that constraint *lin* is present as a tight equation in system $\mathcal{S}^*$. If we skip the equation corresponding to constraint *lin* from $\mathcal{S}^*$, we are remaining with a new system, say $\mathcal{S}^*_1$. Since $\mathcal{S}^*_1$ has $m - 1$ linearly independent equations, it defines a 1-dimensional affine subspace, say F, of the polytope Π'. We know that $x^* \in F$. Therefore, x^* belongs to the intersection of F and the hyperplane defined by constraint *lin*. Since F is a 1-dimensional face of polytope Π', we can decompose x^* into a convex combination of at most 2 extreme points of polytope Π' (this fact basically follows from the theorem of Carathéodory, and the fact that any point in a polytope is a convex combination of its vertices, see [11]). If constraint *lin* is not present as a tight equation in system $\mathcal{S}^*$, then x^* is an extreme point of the polytope Π' itself. Thus, we have shown the following lemma.

Lemma 2. *Any extreme point x^* of the EC-SNDP polytope defined by a weakly supermodular function f with an additional linear constraint lin can be expressed as a convex combination of at most 2 extreme points of the EC-SNDP polytope without constraint lin.*

But we know by the result of Jain [4], that any extreme point of polytope Π' has at least one entry of at least $1/2$. We will use this fact to proceed. By

Lemma 2, $x^* = \lambda x^1 + (1-\lambda)x^2$, where $\lambda \in [0,1]$, and x^1, x^2 are extreme points of the polytope Π'. Then, either $\lambda \geq 1/2$ or $1-\lambda \geq 1/2$. Suppose without loss of generality that $\lambda \geq 1/2$. From the result of Jain mentioned above, we know that there exists an $e \in E$, with $x^1_e \geq 1/2$. Therefore, we obtain that

$$x^*_e = \lambda x^1_e + (1-\lambda)x^2_e \geq \lambda x^1_e \geq 1/4.$$

This concludes the proof of Theorem 1. □

2.2 An Improved Polyhedral Theorem

We will now show how to improve the result of Theorem 1 by exploring the combinatorial structure of the Constr EC-SNDP polytope.

Theorem 2. *Let x^* be any extreme point of the EC-SNDP polytope defined by a weakly supermodular function f with an additional linear constraint lin. Then there is an index $e \in E$ such that $x^*_e \geq 1/3$.*

Proof. This proof builds on some arguments that have been already used by Jain [4]. We assume w.l.o.g. that for any $e \in E$, $0 < x^*_e < 1$. Otherwise, we either can project the polytope onto an $x^*_e = 0$ or we are done if $x^*_e = 1$. Let $\mathcal{S}^*$ be a system defining the extreme point x^*, see Lemma 1. If constraint lin is not tight, then x^* is an extreme point of the EC-SNDP polytope, so by result of Jain [4], there exists an $e \in E$ such that $x^*_e \geq 1/2$, and we are done.

Assume constraint lin is tight for x^* and it is present as a tight equation in the system $\mathcal{S}^*$. Let $\mathcal{T}$ be a family of all tight sets for x^*, i.e.,

$$\mathcal{T} = \{S \subset V : x^*(\delta(S)) = f(S)\}.$$

Let $\mathcal{L} \subseteq \mathcal{T}$ be a maximal laminar subfamily of $\mathcal{T}$. By Lemma 4.2 in [4], $sp(\mathcal{L}) = sp(\mathcal{T})$. We claim that the vector space $sp(\mathcal{T})$ has dimension $\geq |E| - 1 = m - 1$. This is clear, since

$$dim(sp(\mathcal{T}) \cup \boldsymbol{x}_{lin}) = m,$$

where $\boldsymbol{x}_{lin}$ is the vector of the coefficients of the left-hand-side of lin. If now $dim(sp(\mathcal{T})) = m$ then by Jain's [4] theorem there is an e with $x^*_e \geq 1/2$, so the theorem holds. So assume $dim(sp(\mathcal{T})) = m-1$. Since $sp(\mathcal{L}) = sp(\mathcal{T})$, there is a basis $\{\boldsymbol{x}(S) : S \in \mathcal{B}\}$ of the vector space $sp(\mathcal{T})$, with $\mathcal{B} \subseteq \mathcal{L}$ and $|\mathcal{B}| = m-1$. We also notice that for each $S \in \mathcal{B}$ we have $f(S) \geq 1$. This fact follows by a contradiction. Namely, suppose that $f(S) < 0$, then S cannot be tight; also if $f(S) = 0$, then $\boldsymbol{x}(S) = \mathbf{0}$, which gives a contradiction to linear independence. The reasoning above leads therefore to the following lemma.

Lemma 3. *There is a laminar family $\mathcal{B}$ of tight sets, such that $|\mathcal{B}| = m-1$, vectors $\boldsymbol{x}(S)$, $S \in \mathcal{B}$ are linearly independent, and $\forall S \in \mathcal{B}$, $f(S) \geq 1$.*

We represent the laminar family $\mathcal{B}$ as a directed forest as follows. The node set of the forest is $\mathcal{B}$, and there is an edge from $C \in \mathcal{B}$ to $R \in \mathcal{B}$ if R is the smallest set containing C. We say that R is a parent of C and C is a child of R. A parent-less node is called a root, and a child-less node is called a leaf. We prove now the following lemma that will imply the theorem.

Lemma 4. *Laminar family $\mathcal{B}$ contains at least one set S with $|\delta(S)| \leq 3$.*

Proof. Assume towards a contradiction that for any set $S \in \mathcal{B}$ we have $|\delta(S)| \geq 4$. We show a contradiction by distributing the $2|E| = 2m$ endpoints to the $m-1$ nodes of the forest representing the laminar family $\mathcal{B}$. In this distribution, each internal node of a subtree gets at least 2 endpoints, and the root gets at least 4 endpoints. This is true for all subtrees of the forest that are leaves: each such leaf S is crossed by at least 4 edges. Induction step is the same as in the proof of Lemma 4.5 in [4], and we present it here for completeness.

Consider a subtree rooted at R. Suppose first R has two or more children. By the induction hypothesis, each of those children gets at least 4 endpoints, and each of their descendants gets at least 2 endpoints. We will now redistribute these endpoints. Root R borrows two endpoints from each of its children – thus getting at least 4 endpoints. Hence, each descendant of R will still have at least 2 endpoints, and we are done in this case. Assume now, that R has exactly one child, say C. By the induction hypothesis, C gets at least 4 endpoints, and each of its descendants gets at least 2 endpoints. We now borrow two endpoints from C and assign them to R. If R had two more endpoints on its own, i.e., endpoints which were incident to R, the induction step follows. Thus, the only case that is left is to assume that R has at most one endpoint incident to it.

Since $\boldsymbol{x}(R)$ and $\boldsymbol{x}(C)$ are two distinct vectors, there is at least one edge which crosses C but not R, or else crosses R but not C. In both cases there is an endpoint incident to R. By our assumption R has at most one endpoint incident to it, and so R has exactly one endpoint incident to it.

The value x_e of the edge giving one endpoint incident to R is the difference between the requirements of R and C. This is an integer, but by our assumption x_e is strictly fractional – a contradiction. This proves the induction step.

If the forest has at least two roots, we get a contradiction, since the number of endpoints is $2m$, but by the distribution it is at least $2(m-1)+4 = 2m+2$.

Thus, we assume that the forest has exactly one root. In this case the forest is just a tree. We know that after the distribution the (unique) root of the tree has got 4 endpoints. We observe that the root corresponds to a set R of vertices of the original graph, such that R contains every other set in the laminar family $\mathcal{B}$ (by the uniqueness of the root). But by our assumption R must be crossed by at least 4 edges. This means that there are at least 4 endpoints (corresponding to the ends of these 4 edges that are outside R), and we know that these four endpoints were not considered so far in the distribution (since they are not contained in the tree). Therefore we can assign these 4 additional endpoints to the root R. Thus the root gets at least 8 endpoints. This means that the number of endpoints is at least $2(m-1)+6 = 2m+4$. Contradiction. This finishes the proof of Lemma 4, and the proof of Theorem 2, as well. □

3 The Algorithm

Let $G = (V, E)$ be a given undirected multigraph, with a cost function $c : E \to \Re_+$ and a length function $l : E \to \Re_+$. Let L be a given positive number. Assume

that problem $IP(\Pi)$ is to find a minimum-cost (with respect to c) subgraph G' of G such that (1) and (2) hold, and cost of G' with respect to l is at most L. Let $\Pi = \Pi(E)$ correspond to constraints (1)–(2). We can formulate $IP(\Pi)$ as:

$$\begin{aligned} \min \quad & \textstyle\sum_{e\in E} c_e x_e && (3)\\ s.t. \quad & x \in \Pi(E) && (4)\\ & \textstyle\sum_{e\in E} l_e x_e \le L && (5)\\ & x_e \in \{0,1\} \quad \forall e \in E. && (6) \end{aligned}$$

Let (LP) denote the LP relaxation of $IP(\Pi)$, that is, (LP) corresponds to (3)–(6) with (6) replaced by $x_e \in [0,1]$ $\forall e \in E$.

The algorithm is recursive. In the first step it solves the (LP) optimally, producing a basic solution $\bar{x}$ to (LP). By Theorem 2 there exists $e \in E$ such that $\bar{x}_e \ge \frac{1}{3}$. Let $E(\frac{1}{3}) = \{e \in E : \bar{x}_e \ge \frac{1}{3}\}$, and opt_{LP} denote the value of an optimal fractional solution to (LP).

The algorithm rounds each $\bar{x}_e$, for $e \in E(\frac{1}{3})$, to one: $E(\frac{1}{3})$ is a part of the solution. Then the algorithm recursively solves the following integer program, called (IP′).

$$\begin{aligned} \min \quad & \textstyle\sum_{e\in E\setminus E(\frac{1}{3})} c_e x_e\\ s.t. \quad & x \in \Pi(E \setminus E(\tfrac{1}{3}))\\ & \textstyle\sum_{e\in E\setminus E(\frac{1}{3})} l_e x_e \le L - \sum_{e\in E(\frac{1}{3})} l_e \bar{x}_e\\ & x_e \in \{0,1\} \quad \forall e \in E \setminus E(\tfrac{1}{3}), \end{aligned}$$

where $\Pi(E \setminus E(\frac{1}{3}))$ corresponds to constraints (1)–(2) with E replaced by $E \setminus E(\frac{1}{3})$, and constraints (1) replaced by

$$\sum_{e\in\delta_{G'}(S)} x_e \ge f(S) - |E(1/3) \cap \delta_G(S)| \quad \forall S \subseteq V, \quad \text{where} \quad G' = (V, E \setminus E(1/3)).$$

We summarize our algorithm for Constr EC-SNDP in Figure 1 below.

Algorithm **IterativeRound**(E, L)**:**
1 **if** E has constant size **then** solve E optimally and output its solution;
2 solve (LP) optimally; let $\bar{x} \in [0,1]^{|E|}$ be the resulting basic solution;
3 $E(\frac{1}{3}) := \{e \in E : \bar{x}_e \ge \frac{1}{3}\}$;
4 output $E(\frac{1}{3}) \cup$ **IterativeRound**$(E \setminus E(\frac{1}{3}),\ L - \sum_{e\in E(\frac{1}{3})} l_e \bar{x}_e)$.

Fig. 1. The recursive iterative rounding algorithm for Constr EC-SNDP.

We would like to note here that if we use in the algorithm above Theorem 1 instead of Theorem 2 this would obviously lead to a weaker approximation algorithm for Constr EC-SNDP. However, the arguments in the proof of Theorem 1 may prove useful – see Section 5.

4 The Analysis

We will first prove the following fact about all linear programs that may appear during the execution of the algorithm.

Lemma 5. *All the linear programs of form (LP) as defined in Section 3 that the algorithm IterativeRound will meet during its execution are fractionally feasible.*

Proof. The proof is by induction on the iterations of the algorithm. Let us consider the first iteration, in which the algorithm solves exactly (LP) as above. Then, since our original integral problem Constr EC-SNDP has an integral feasible solution, this solution is also feasible for the (LP).

We will now show the induction step. Let (LP′) be the LP relaxation of (IP′), obtained in the same way as (LP). Suppose now that (LP) was a linear program met by our algorithm in some iteration, and let (LP′) be the linear program met in the very next iteration. By induction assumption (LP) is fractionally feasible, and let $\bar{x}$ be an optimal basic fractional solution to (LP). The restriction of $\bar{x}$ to $E \setminus E(\frac{1}{3})$ is a feasible solution to (LP′), and so the claim holds. □

We are now ready to prove the main theorem in this paper.

Theorem 3. *The algorithm IterativeRound defined in Section 3 is a strongly polynomial time* $(3,3)$*-approximation algorithm for the* Constr EC-SNDP *problem.*

Proof. We will prove the following statement: if a linear program (LP) as defined in Section 3 is met by the algorithm at some iteration and this linear program is fractionally feasible, then our algorithm outputs a corresponding integral solution to this (LP) which is within a factor of 3 of the optimal fractional solution to this (LP), and the length of this integral solution is within a factor of 3 of the budget in this (LP). We will prove this statement by induction on the number of iterations of our algorithm.

Suppose first that an instance of Constr EC-SNDP requires only one iteration of the algorithm. By Lemma 5, the linear program involved, say (LP), is fractionally feasible. Thus, the algorithm finds an optimum basic solution to (LP), say $\bar{x}$, and rounds up all edges e with $\bar{x}_e \geq 1/3$. Thus, $E(\frac{1}{3})$ is an integral feasible solution to (LP), and we will argue that it gives a $(3,3)$-approximation. Since for each $e \in E(\frac{1}{3})$, $3\bar{x}_e \geq 1$, we have that

$$\sum_{e \in E(\frac{1}{3})} c_e \leq 3 \cdot opt_{LP}.$$

Similarly, we can argue that

$$\sum_{e \in E(\frac{1}{3})} l_e \leq \sum_{e \in E(\frac{1}{3})} 3\bar{x}_e l_e \leq 3 \sum_{e \in E} \bar{x}_e l_e \leq 3L.$$

We will now show the induction step. Observe, that the LP which is the relaxation of (IP′) still fulfills Theorem 2 (this follows from Theorem 2.5 in

[4]). Thus, (IP′) is an instance of the same form as (IP). Let (LP′) be the LP relaxation to (IP′), obtained in the same way as (LP). We now argue about the approximation ratio. By Lemma 5 the linear program (LP′) is fractionally feasible, and so by induction assumption there is an integral solution, say E', to (LP′) with cost at most $3 \cdot opt_{LP'}$ and length at most

$$3 \cdot \left(L - \sum_{e \in E(1/3)} l_e \bar{x}_e \right).$$

We show that $E(\frac{1}{3}) \cup E'$ is an integral solution to (LP) with cost at most $3 \cdot opt_{LP}$ and length at most $3L$.

The restriction of $\bar{x}$ to $E \setminus E(\frac{1}{3})$ is a feasible solution to (LP′), so

$$opt_{LP'} \leq opt_{LP} - \sum_{e \in E(\frac{1}{3})} c_e \bar{x}_e.$$

But for each $e \in E(\frac{1}{3})$, $3\bar{x}_e \geq 1$, hence

$$3 \cdot opt_{LP'} + \sum_{e \in E(\frac{1}{3})} c_e \leq 3 \cdot opt_{LP}.$$

From the induction assumption, $\sum_{e \in E'} c_e \leq 3 \cdot opt_{LP'}$, thus

$$\sum_{e \in E'} c_e + \sum_{e \in E(\frac{1}{3})} c_e \leq 3 \cdot opt_{LP}.$$

Also by the induction assumption, $\sum_{e \in E'} l_e \leq 3 \cdot (L - \sum_{e \in E(\frac{1}{3})} l_e \bar{x}_e)$. Therefore, the length of the output solution is

$$\sum_{e \in E'} l_e + \sum_{e \in E(\frac{1}{3})} l_e \leq \sum_{e \in E'} l_e + 3 \cdot \sum_{e \in E(\frac{1}{3})} l_e \bar{x}_e \leq 3 \cdot L.$$

Jain [4] has shown that LPs (LP) corresponding to our original problem without the budget constraint can be solved in strongly polynomial time. Norton et al. [7] prove that if we have a strongly polynomial time algorithm for a linear program, then this linear program can still be solved in strongly polynomial time when a constant number of linear constraints are added to the linear program. This gives a strongly polynomial algorithm for our problem. (The approach in [7] requires in fact a specific strongly polynomial time algorithm, called "linear", and [4] indeed provides such an algorithm.)

We will finally note that we could also use Theorem 1 instead of Theorem 2 to obtain a strongly polynomial time $(4, 4)$-approximation algorithm for the Constr EC-SNDP problem. □

5 Conclusions and Further Questions

We have obtained in this paper a strongly polynomial time $(3, 3)$-approximation algorithm for the Constr EC-SNDP problem, by extending the iterative rounding technique. For this result we needed explore the combinatorial structure of the Constr EC-SNDP polytope in Section 2.2.

We have also shown in Section 2.1 that a straightforward polyhedral argument implies a weaker strongly polynomial time $(4, 4)$-approximation algorithm for this problem. This second polyhedral argument might be interesting for the following reason. Suppose, that we are given a problem which can be formulated as an integer linear program. Let us now take an LP relaxation to this integer linear program, and suppose that we are able to prove that any two adjacent extreme points of this relaxation have two large entries (maybe even at the same vector position). Then, we will be able to show that an extreme point of this polytope with an additional linear constraint also has a large entry – see the argument at the end of the proof of Theorem 1. Similar argument may be applied in case where we add more than just one additional linear constraint. We leave finding such applications as an open problem.

References

1. A. Agrawal, P. Klein and R. Ravi. When trees collide : An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing*, **24**(3), pp. 445–456, 1995.
2. J.L. Ganley, M.J. Golin and J.S. Salowe. The Mutli-Weighted Spanning Tree Problem. In the *Proc. 1st Conference on Combinatorics and Computing (COCOON)*, Springer Verlag, LNCS, pp. 141–150, 1995.
3. M.X. Goemans and D.P. Williamson. A General Approximation Technique for Constrained Forest Problems. *SIAM Journal on Computing*, **24**, pp. 296–317, 1995.
4. K. Jain. A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem. *Combinatorica*, **21**(1), pp. 39–60, 2001. Available at http://www.cc.gatech.edu/people/home/kjain/
5. S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, **41**, pp. 214–235, 1994.
6. M.V. Marathe, R. Ravi, R. Sundaram, S.S. Ravi, D.J. Rosenkrantz and H.B. Hunt III. Bicriteria Network Design. *Journal of Algorithms*, **28**, pp. 142–171, 1998.
7. C.H. Norton, S.A. Plotkin and É. Tardos. Using Separation Algorithms in Fixed Dimension. *Journal of Algorithms*, **13**, pp. 79–98, 1992.
8. R. Ravi and M.X. Goemans. The Constrained Minimum Spanning Tree Problem. In the *Proc. of the Scandinavian Workshop of Algorithmic Theory (SWAT)*, Springer Verlag, LNCS **1097**, pp. 66–75, 1996.
9. R. Ravi, M.V. Marathe, S.S. Ravi, D.J. Rosenkrantz and H.B. Hunt III. Many birds with one stone: multi-objective approximation algorithms. In the *Proc. 25th ACM Symposium on Theory of Computing (STOC)*, 1993.
10. G. Robins and A. Zelikovsky. Improved Steiner Tree Approximation in Graphs. In the *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 770–779, 2000.
11. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, 1986.

Interference in Cellular Networks: The Minimum Membership Set Cover Problem

Fabian Kuhn[1], Pascal von Rickenbach[1], Roger Wattenhofer[1], Emo Welzl[2], and Aaron Zollinger[1]

[1] Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich, Switzerland
{kuhn,pascalv,wattenhofer,zollinger}@tik.ee.ethz.ch
[2] Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland
welzl@inf.ethz.ch

Abstract. The infrastructure for mobile distributed tasks is often formed by cellular networks. One of the major issues in such networks is interference. In this paper we tackle interference reduction by suitable assignment of transmission power levels to base stations. This task is formalized introducing the *Minimum Membership Set Cover* combinatorial optimization problem. On the one hand we prove that in polynomial time the optimal solution of the problem cannot be approximated more closely than with a factor $\ln n$. On the other hand we present an algorithm exploiting linear programming relaxation techniques which asymptotically matches this lower bound.

1 Introduction

Cellular networks are heterogeneous networks consisting of two different types of nodes: base stations and clients. The base stations – acting as servers – are interconnected by an external fixed backbone network; clients are connected via radio links to base stations. The totality of the base stations forms the infrastructure for distributed applications running on the clients, the most prominent of which probably being mobile telephony. Cellular networks can however more broadly be considered a type of infrastructure for mobile distributed tasks in general.

Since communication over the wireless links takes place in a shared medium, interference can occur at a client if it is within transmission range of more than one base station. In order to prevent such collisions, coordination among the conflicting base stations is required. Commonly this problem is solved by segmenting the available frequency spectrum into channels to be assigned to the base stations in such a way as to prevent interference, in particular such that no two base stations with overlapping transmission range use the same channel.

In this paper we assume a different approach to interference reduction. The basis of our analysis is formed by the observation that interference effects occurring at a client depend on the number of base stations by whose transmission ranges it is covered. In particular for solutions using frequency division multiplexing as described above, the number of base stations covering a client is a lower bound for the number of channels required to avoid conflicts; a reduction in the required number of channels, in turn, can be exploited to broaden the frequency segments and consequently to increase communication bandwidth. On the other hand, also with systems using code division multiplexing,

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 188–198, 2005.

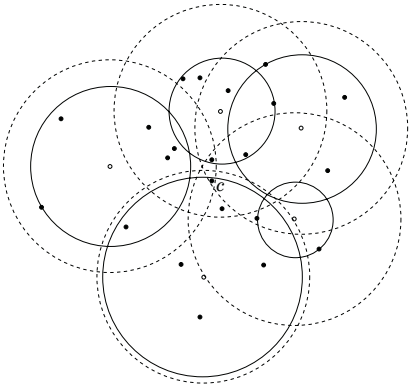


Fig. 1. If the base stations (hollow points) are assigned identical transmission power levels (dashed circles), client c experiences high interference, since it is covered by all base stations. Interference can be reduced by assigning appropriate power values (solid circles), such that all clients are covered by at most two base stations.

the coding overhead can be reduced if only a small number of base stations cover a client.

The transmission range of a base station – and consequently the coverage properties of the clients – depends on its position, obstacles hindering the propagation of electromagnetic waves, such as walls, buildings, or mountains, and the base station transmission power. Since due to legal or architectural constraints the former two factors are generally difficult to control, we assume a scenario in which the base station positions are fixed, each base station can however adjust its transmission power. The problem of minimizing interference then consists in assigning every base station a transmission power level such that the number of base stations covering any node is minimal (cf. Figure 1). At the same time however, it has to be guaranteed that every client is covered by at least one base station in order to maintain availability of the network.

In Figure 1 the area covered by a base station b transmitting with a given power level is represented by a disk centered at b and having a radius corresponding to the chosen transmission power. Practical measurements however show that this idealization is far from realistic. Not only mechanical and electronical inaccuracies inevitable in the construction of antennas, but more importantly the presence of obstacles to the propagation of electromagnetic signals – such as buildings, mountains, or even weather conditions – can lead to areas covered by signal transmission that hardly resemble disks in practice. These considerations motivate that in order to study the described interference reduction problem we abstract from network node positions and circular transmission areas.

In our analysis we formalize the task of reducing interference as a combinatorial optimization problem. For this purpose we model the transmission range of a base station having chosen a specific transmission power level as a set containing exactly all clients covered thereby. The totality of transmission ranges selectable by all base stations is consequently modeled as a collection of client sets. More formally, this yields the *Minimum Membership Set Cover (MMSC)* problem: Given a set of elements U (modeling clients) and a collection S of subsets of U (transmission ranges), choose a solution $S' \subseteq S$ such that every element occurs in at least one set in S' (maintain network

availability) and that the *membership* $M(u, S')$ of any element u with respect to S' is minimal, where $M(u, S')$ is defined as the number of sets in S' in which u occurs (interference)[1].

Having defined this formalization, we show in this paper – by reduction from the related Minimum Set Cover problem – that the MMSC problem is NP-complete and that no polynomial time algorithm exists with approximation ratio less than $\ln n$ unless $NP \subset TIME(n^{O(\log \log n)})$. In a second part of the paper we present a probabilistic algorithm based on linear programming relaxation and derandomization asymptotically matching this lower bound, particularly yielding an approximation ratio in $O(\ln n)$.

The paper is organized as follows: Discussing related work in Section 2, we formally define the MMSC problem in Section 3. Section 4 contains a description of the lower bound with respect to approximability of the MMSC problem. In the subsequent section we describe how the MMSC problem can be formulated as a linear program and provide a $O(\ln n)$-approximation algorithm for the problem. Section 6 concludes the paper.

2 Related Work

Interference issues in cellular networks have been studied since the early 1980s in the context of frequency division multiplexing: The available network frequency spectrum is divided into narrow channels assigned to cells in a way to avoid interference conflicts. In particular two types of conflicts can occur, adjacent cells using the same channel (cochannel interference) and insufficient frequency distance between channels used within the same cell (adjacent channel interference). Maximizing the reuse of channels respecting these conflicts was generally studied by means of the combinatorial problem of conflict graph coloring using a minimum number of colors. The settings in which this problem was considered are numerous and include hexagon graphs, geometric intersection graphs (such as unit disk graphs), and planar graphs, but also (non-geometric) general graphs. In addition both static and dynamic (or on-line) approaches were studied [11]. The fact that channel separation constraints can depend on the distance of cells in the conflict graph was analyzed by means of graph labeling [6]. The problem of frequency assignment was tackled in a different way in [2] exploiting the observation that in every region of an area covered by the communication network it is sufficient that exactly one base station with a unique channel can be heard. As mentioned, all these studied models try to avoid interference conflicts occurring when using frequency division multiplexing. In contrast, the problem described in this paper assumes a different approach in aiming at interference reduction by having the base stations choose suitable transmission power levels.

The problem of reducing interference is formalized in a combinatorial optimization problem named *Minimum Membership Set Cover*. As suggested by its name, at first sight its formulation resembles closely the long-known and well-studied *Minimum Set Cover (MSC)* problem, where the number of sets chosen to cover the given elements is to be minimized [7]. That the MMSC and the MSC problems are however of different nature can be concluded from the following observation: For any MSC instance

[1] Note that naturally, for each base station, the client set corresponding to a particular power level contains all sets corresponding to lower power levels. Thus, we can assume that w.l.o.g., only one client set is chosen for each base station.

consisting of n elements, a greedy algorithm approximates the optimal solution with an approximation ratio at most $H(n) \leq \ln n + 1$ [7], which has later been shown to be tight up to lower order terms unless $NP \subset TIME(n^{O(\log \log n)})$ [3, 10]. For the MMSC problem in contrast, there exist instances where the same greedy algorithm fails to achieve *any* nontrivial approximation of the optimal solution.

The approximation algorithm for the MMSC problem introduced in this paper is based on the formulation of a given instance as a linear program. Solving this linear program yields values subsequently interpreted as probabilities with which to randomly decide for every set in S whether it should belong to the solution. This technique, commonly known as randomized rounding was proposed in [12]. Also derandomization based on the method of conditional probabilities – the technique exploited to transform the above probabilistic algorithm into a deterministic one – was introduced in [12] and extended as well as improved in [13].

In the context of network traffic congestion, [9] considered a problem similar to our analysis of the MMSC problem in that linear program relaxation was employed to minimize a maximum value.

3 Minimum Membership Set Cover

As described in the introduction, the problem considered in this paper is to assign to each base station a transmission power level such that interference is minimized while all clients are covered. For our analysis we formalize this problem by introducing a combinatorial optimization problem referred to as *Minimum Membership Set Cover*. In particular, clients are modeled as elements and the transmission range of a base station given a certain power level is represented as the set of thereby covered elements. In the following, we first define the membership of an element given a collection of sets:

Definition 1 (Membership) *Let U be a finite set of elements and S be a collection of subsets of U. Then the membership $M(u, S)$ of an element u is defined as $|\{T \mid u \in T, T \in S\}|$.*

Informally speaking, MMSC is identical to the MSC problem apart from the minimization function. Where MSC minimizes the total number of sets, MMSC tries to minimize element membership. Particularly, MMSC can be defined as follows:

Definition 2 (Minimum Membership Set Cover) *Let U be a finite set of elements with $|U| = n$. Furthermore let $S = \{S_1, \ldots, S_m\}$ be a collection of subsets of U such that $\bigcup_{i=1}^{m} S_i = U$. Then Minimum Membership Set Cover (MMSC) is the problem of covering all elements in U with a subset $S' \subseteq S$ such that $\max_{u \in U} M(u, S')$ is minimal*[2].

[2] Besides minimizing the *maximal* membership value over all elements, also minimization of the *average* membership value can be considered a reasonable characterization of the interference reduction problem. The fact however that – given a solution S' – the sum of all membership values equals the sum of the cardinalities of the sets in S' shows that this min-average variant is identical to the Weighted Set Cover [1] problem with the set weights corresponding to their cardinalities.

Note that – as motivated in the introduction – the problem statement does not require the collection of subsets S to reflect geometric positions of network nodes. For a given problem instance to be valid, $\bigcup_{i=1}^{m} S_i = U$ is sufficient.

4 Problem Complexity

In this section we address the complexity of the *Minimum Membership Set Cover* problem. We show that MMSC is NP-complete and therefore no polynomial time algorithm exists that solves MMSC unless $P = NP$.

Theorem 1. *MMSC is NP-complete.*

Proof. We will prove that MMSC is NP-complete by reducing MSC to MMSC. Consider an MSC instance (U, S) consisting of a finite set of elements U and a collection S of subsets of U. The objective is to choose a subset S' with minimum cardinality from S such that the union of the chosen subsets of U contains all elements in U.

We now define a set $\widetilde{U}$ by adding a new element e to U, construct a new collection of sets $\widetilde{S}$ by inserting e into all sets in S, and consider $(\widetilde{U}, \widetilde{S})$ as an instance of MMSC. Since element e is in every set in $\widetilde{S}$, it follows that e is an element with maximum membership in the solution S' of MMSC. Moreover, the membership of e in S' is equal to the number of sets in the solution. Therefore MMSC minimizes the number of sets in the solution by minimizing the membership of e. Consequently we obtain the solution for MSC of the instance (U, S) by solving MMSC for the instance $(\widetilde{U}, \widetilde{S})$ and extracting element e from all sets in the solution.

We have shown a reduction from MSC to MMSC, and therefore the latter is NP-hard. Since solutions for the decision problem of MMSC are verifiable in polynomial time, it is in NP, and consequently the MMSC decision problem is also NP-complete. □

Now that we have proved MMSC to be NP-complete and therefore not to be optimally computable within polynomial time unless $P = NP$, the question arises, how closely MMSC can be approximated by a polynomial time algorithm. This is partly answered with the following lower bound.

Theorem 2. *There exists no polynomial time approximation algorithm for MMSC with an approximation ratio less than $(1 - \mathrm{o}(1)) \ln n$ unless $NP \subset TIME(n^{\mathrm{O}(\log \log n)})$.*

Proof. The reduction from MSC to MMSC in the proof of Theorem 1 is approximation-preserving, that is, it implies that any lower bound for MSC also holds for MMSC. In [3] it is shown that $\ln n$ is a lower bound for the approximation ratio of MSC unless $NP \subset TIME(n^{\mathrm{O}(\log \log n)})$. Thus, $\ln n$ is also a lower bound for the approximation ratio of MMSC. □

5 Approximating MMSC by LP Relaxation

In the previous section a lower bound of $\ln n$ for the approximability of the MMSC problem by means of polynomial time approximation algorithms has been established. In this section we show how to obtain a $\mathrm{O}(\log n)$-approximation using LP relaxation techniques.

5.1 LP Formulation of MMSC

We first derive the integer linear program which describes the MMSC problem. Let $S' \subseteq S$ denote a subset of the collection S. To each $S_i \in S$ we assign a variable $x_i \in \{0,1\}$ such that $x_i = 1 \Leftrightarrow S_i \in S'$. For S' to be a set cover, it is required that for each element $u_i \in U$, at least one set S_j with $u_i \in S_j$ is in S'. Therefore, S' is a set cover of U if and only if for all $i = 1, ..., n$ it holds that $\sum_{S_j \ni u_i} x_j \geq 1$. For S' to be minimal in the number of sets that cover a particular element, we need a second set of constraints. Let z be the maximum membership over all elements caused by the sets in S'. Then for all $i = 1, ..., n$ it follows that $\sum_{S_j \ni u_i} x_j \leq z$. The MMSC problem can consequently be formulated as the integer program $\mathrm{IP}_{\mathrm{MMSC}}$:

$$
\begin{aligned}
&\text{minimize } z \\
&\text{subject to} \quad \sum_{S_j \ni u_i} x_j \geq 1 \qquad i = 1, ..., n \\
&\qquad\qquad \sum_{S_j \ni u_i} x_j \leq z \qquad i = 1, ..., n \\
&\qquad\qquad x_j \in \{0,1\} \qquad j = 1, ..., m
\end{aligned}
$$

By relaxing the constraints $x_j \in \{0,1\}$ to $x'_j \geq 0$, we obtain the linear program $\mathrm{LP}_{\mathrm{MMSC}}$. The integer program $\mathrm{IP}_{\mathrm{MMSC}}$ yields the optimal solution z^* for an MMSC problem. The linear program LP_{MMSC} therefore results in a fractional solution z' with $z' \leq z^*$, since we allow the variables x'_j to be in [0,1].

5.2 Randomized Rounding

In [12] and [13], randomized rounding was introduced for covering and packing problems. In the following, we show that this technique can also be applied to solve $\mathrm{IP}_{\mathrm{MMSC}}$ resulting in an almost optimal algorithm. We present an efficient $(1+\mathrm{O}(1/\sqrt{z'}))(\ln(n) + 1)$-approximation algorithm for the MMSC problem. Given an MMSC instance (U, S), we first solve the linear program $\mathrm{LP}_{\mathrm{MMSC}}$ corresponding to (U, S), yielding a vector $\underline{x'}$ and z' and then apply randomized rounding in order to obtain an integer solution. Consider the following "simple" randomized rounding scheme. We compute an integer solution $\underline{x} \in \{0,1\}^m$ by setting

$$
x_i := \begin{cases} 1 & \text{with probability } p_i := \min\{1, \alpha x'_i\} \text{ for a value } \alpha \geq 1 \\ 0 & \text{otherwise} \end{cases}
$$

independently for each $i \in \{1, \ldots, m\}$. Let $\mathcal{A}_i$ be the *bad* event that the i^{th} element is *not* covered.

Lemma 1. *The probability that the i^{th} element remains uncovered is*

$$
\mathbb{P}(\mathcal{A}_i) \;=\; \prod_{S_j \ni u_i} (1 - p_j) \;< e^{-\alpha}.
$$

Proof. Let m_i be the number of sets containing element i ($m_i = |\{j|S_j \ni u_i\}|$). By the "means inequality", we have

$$\mathbb{P}(\mathcal{A}_i) = \prod_{S_j \ni u_i} (1-p_j) \leq \left(1 - \frac{\sum_{S_j \ni u_i} p_j}{m_i}\right)^{m_i} \leq \left(1 - \frac{\alpha}{m_i}\right)^{m_i} < e^{-\alpha}.$$

Note that $\sum_{S_j \ni u_i} p_j \geq \alpha$ only holds if all $p_j < 1$. We can safely make this assumption because $p_j = 1$ for some $S_j \ni u_i$ makes $\mathbb{P}(\mathcal{A}_i) = 0$. □

Let $\mathcal{B}_i$ be the *bad* event that the i^{th} element is covered by *more* than $\alpha\beta z'$ sets for some $\beta \geq 1$.

Lemma 2. *The probability that the i^{th} element is covered more than $\alpha\beta z'$ times is*

$$\mathbb{P}(\mathcal{B}_i) < \frac{1}{\beta^{\alpha\beta z'}} \cdot \prod_{S_j \ni u_i} [1 + (\beta-1)p_j] \leq \left(\frac{e^{\beta-1}}{\beta^\beta}\right)^{\alpha z'}.$$

Proof. We use a Chernoff-type argument. For $t = \ln\beta > 0$, we have

$$\begin{aligned}
\mathbb{P}(\mathcal{B}_i) &= \mathbb{P}\left(\sum_{S_j \ni u_i} x_j > \alpha\beta z'\right) = \mathbb{P}\left(e^{t\cdot\sum_{S_j \ni u_i} x_j} > e^{t\alpha\beta z'}\right) \\
&< \frac{\mathrm{E}\left[e^{t\cdot\sum_{S_j \ni u_i} x_j}\right]}{e^{t\alpha\beta z'}} = \frac{1}{e^{t\alpha\beta z'}} \cdot \prod_{S_j \ni u_i} \left[p_j e^t + 1 - p_j\right] \\
&= \frac{1}{\beta^{\alpha\beta z'}} \cdot \prod_{S_j \ni u_i} [1 + (\beta-1)p_j] \leq \frac{1}{\beta^{\alpha\beta z'}} \cdot \prod_{S_j \ni u_i} e^{(\beta-1)p_j} \leq \left(\frac{e^{\beta-1}}{\beta^\beta}\right)^{\alpha z'}.
\end{aligned}$$

The inequality in the first line results by application of the Markov inequality. The equations in the second line hold because of the independence of the x_i and because $t = \ln\beta$, respectively. For the inequalities in the last line, we apply $1 + x \leq e^x$ and $\sum_{S_j \ni u_i} p_j \leq \alpha z'$. □

In the following, we denote the probability upper bounds given by Lemmas 1 and 2 by A_i and B_i:

$$A_i := \prod_{S_j \ni u_i} (1-p_j) \quad \text{and} \quad B_i := \frac{1}{\beta^{\alpha\beta z'}} \cdot \prod_{S_j \ni u_i} [1 + (\beta-1)p_j].$$

In order to bound the probability for any bad event to occur, we define a function P as follows:

$$P(p_1, \ldots, p_m) := 2 - \prod_{i=1}^{n}(1 - A_i) - \prod_{i=1}^{n}(1 - B_i).$$

Lemma 3. *The probability that any element is not covered or covered more than $\alpha\beta z'$ times is upper-bounded by $P(p_1, \ldots, p_m)$:*

$$\mathbb{P}\left(\bigcup_{i=1}^{n} \mathcal{A}_i \cup \bigcup_{i=1}^{n} \mathcal{B}_i\right) < P(p_1, \ldots, p_m).$$

Proof. It is sufficient to prove that

$$\mathbb{P}\left(\bigcup_{i=1}^{n} \mathcal{A}_i\right) \leq 1 - \prod(1 - \mathbb{P}(\mathcal{A}_i)) \quad \text{and} \quad \mathbb{P}\left(\bigcup_{i=1}^{n} \mathcal{B}_i\right) \leq 1 - \prod(1 - \mathbb{P}(\mathcal{B}_i)). \quad (1)$$

The lemma then follows by Lemmas 1 and 2. If the events $\mathcal{A}_i$ and $\mathcal{B}_i$ were independent, the first and second inequality of (1) would hold with equality, respectively. Hence, we have to show that the dependence of the events can only help us. As shown in [13], the, the complementary events $\overline{\mathcal{A}}_i$ are positively correlated, that is, the probability of $\overline{\mathcal{A}}_i$ ($\mathcal{A}_i$ does not occur) increases under the condition that any subset of $\{\overline{\mathcal{A}}_1, \ldots, \overline{\mathcal{A}}_n\}$ occurs. This positive correlation follows from Harris-Kleitman inequality [5, 8], which is a special case of the FKG inequality [4]. Hence, the first inequality of (1) follows. For the events $\mathcal{B}_i$ exactly the same argumentation holds. □

In the following we show that if α and β are chosen appropriately, $P(p_1, \ldots, p_m)$ is always less than 1.

Lemma 4. *When setting $\alpha = \ln(n) + 1$, then for $\beta = 1 + \max\{\sqrt{3/z'}, 3/z'\}$, we have $P(p_1, \ldots, p_m) < 4/5$.*

Proof. By Lemmas 1 and 2, we have

$$P(p_1, \ldots, p_m) < 2 - \left(1 - e^{-\alpha}\right)^n - \left(1 - \left(\frac{e^{\beta-1}}{\beta^\beta}\right)^{\alpha z'}\right)^n.$$

In order to have $P < 4/5$, it therefore suffices to choose α and β such that

$$\left(1 - e^{-\alpha}\right)^n \geq \frac{3}{5} \quad \text{and} \quad \left(1 - \left(\frac{e^{\beta-1}}{\beta^\beta}\right)^{\alpha z'}\right)^n \geq \frac{3}{5}. \quad (2)$$

For $\alpha \geq \ln n + 1$, we get $(1 - e^{-\alpha})^n \geq 3/5$ and therefore the first inequality of (2) is fulfilled. The second inequality of (2) can be transformed into a simpler form using the following inequalities:

$$\frac{e^{\beta-1}}{\beta^\beta} \leq \begin{cases} e^{-(\beta-1)^2/3} & \text{for } 1 \leq \beta \leq 2, \\ e^{-(\beta-1)/3} & \text{for } \beta > 2. \end{cases} \quad (3)$$

If we choose $\beta = 1 + \sqrt{3/z'}$, for $z' \geq 3$, we have $\beta \leq 2$ and therefore by (3), the second inequality of (2) simplifies to

$$\left(1 - \left(\frac{e^{\beta-1}}{\beta^\beta}\right)^{\alpha z'}\right)^n \geq \left(1 - e^{-\alpha z'(\beta-1)^2/3}\right)^n = \left(1 - e^{-\alpha}\right)^n.$$

For $z' < 3$, we can set $\beta = 1 + 3/z' \geq 2$ and proceed analogously using the second case of (3). □

Lemmas 1–4 lead to the following randomized algorithm for the MMSC problem. As a first step, the linear program $\mathrm{LP}_{\mathrm{MMSC}}$ has to be solved. Then, all x'_i are rounded to integer values $x_i \in \{0, 1\}$ using the described randomized rounding scheme with $\alpha = \ln n + 1$. The rounding is repeated until the solution is feasible (all elements are covered) and the membership of the integer solution deviates from the fractional membership z' by at most a factor $\alpha\beta$ for $\beta = 1 + \max\{3/z', \sqrt{3/z'}\}$. Each time, the probability to be successful is at least $1/5$ and therefore, the probability of not being successful decreases exponentially in the number of trials.

5.3 Derandomization

We will now show that $P(p_1, \ldots, p_m)$ is a pessimistic estimator [12, 13] and that therefore, the algorithm described at the end of the previous section can be derandomized. That is, P is an upper bound on the probability of obtaining a "bad" solution, $P < 1$ (P is a probabilistic proof that a "good" solution exists), and the p_i can be set to 0 or 1 without increasing P. The first two properties follow by Lemmas 3 and 4, the third property is shown by the following lemma.

Lemma 5. *For all i, either setting p_i to 0 or setting p_i to 1 does not increase P:*

$$P(p_1, \ldots, p_m) \;\geq\; \min\{P(\ldots, p_{i-1}, 0, p_{i+1}, \ldots), P(\ldots, p_{i-1}, 1, p_{i+1}, \ldots)\}.$$

Proof. We prove the lemma by showing that P is a concave function of p_i:

$$P(p_1, \ldots, p_m) \geq (1-p_i)P(\ldots, p_{i-1}, 0, p_{i+1}, \ldots) + p_i P(\ldots, p_{i-1}, 1, p_{i+1}, \ldots). \quad (4)$$

If all probabilities except p_i are fixed, A_j and B_j are functions of p_i. We define

$$\overline{A}_k(p_i) \;:=\; \prod_{j=1}^{k}(1 - A_j) \quad \text{and} \quad \overline{B}_k(p_i) \;:=\; \prod_{j=1}^{k}(1 - B_j).$$

In order to obtain (4), we prove that

$$\overline{A}_k(p_i) \leq (1-p_i)\overline{A}_k(0) + p_i\overline{A}_k(1) \quad \text{and} \quad \overline{B}_k(p_i) \leq (1-p_i)\overline{B}_k(0) + p_i\overline{B}_k(1) \quad (5)$$

for all $k \in [0, n]$ by induction over k. For $k = 0$, we have $\overline{A}_0(p_i) = \overline{B}_0(p_i) = 1$ and therefore (5) holds. The induction step from k to $k + 1$ depends on whether element $k + 1$ is in set S_i. If element $k + 1$ is not in set S_i, A_{k+1} and B_{k+1} do not depend on p_i and (5) follows from the induction hypothesis. It remains to prove the interesting case where element $k + 1$ is contained in set S_i. We first consider the inequality for $\overline{A}_{k+1}(p_i)$. When p_i is set to 1, A_{k+1} becomes 0. If p_i is set to 0, the factor $1 - p_i$ in A_{k+1} is replaced by 1 and therefore A_{k+1} becomes

$$A_{k+1,p_i=0} = \prod_{S_j \in u_{k+1} \setminus S_i} (1 - p_j) \;=\; \frac{A_{k+1}}{1 - p_i}.$$

We therefore have

$$\begin{aligned}(1-p_i)\overline{A}_{k+1}(0)+p_i\overline{A}_{k+1}(1) &= (1-p_i)\overline{A}_k(0)\cdot\left(1-\frac{A_{k+1}}{1-p_i}\right)+p_i\overline{A}_k(1)\cdot 1\\ &= (1-p_i)\overline{A}_k(0)+p_i\overline{A}_k(1)-\overline{A}_k(0)\cdot A_{k+1}\\ &\geq \overline{A}_k(p_i)(1-A_{k+1}) \;=\; \overline{A}_{k+1}(p_i).\end{aligned}$$

The inequality in the third line follows from the induction hypothesis and from $\overline{A}_k(0) \leq \overline{A}_k(p_i)$. For $\overline{B}_{k+1}(p_i)$, setting p_i to 0 and 1 replaces the factor $1+(\beta-1)p_i$ in B_{k+1} by 1 and β, respectively:

$$B_{k+1,p_i=0}=\frac{B_{k+1}}{1+(\beta-1)p_i} \quad \text{and} \quad B_{k+1,p_i=1}=\frac{\beta B_{k+1}}{1+(\beta-1)p_i}.$$

Thus, we get

$$\begin{aligned}&(1-p_i)\overline{B}_{k+1}(0)+p_i\overline{B}_{k+1}(1)\\ &= (1-p_i)\overline{B}_k(0)\cdot\left(1-\frac{B_{k+1}}{1+(\beta-1)p_i}\right)+p_i\overline{B}_k(1)\cdot\left(1-\frac{\beta B_{k+1}}{1+(\beta-1)p_i}\right)\\ &= \left((1-p_i)\overline{B}_k(0)+p_i\overline{B}_k(1)\right)\cdot\left(1-\frac{B_{k+1}}{1+(\beta-1)p_i}\right)-\frac{B_{k+1}p_i(\beta-1)\overline{B}_k(1)}{1+(\beta-1)p_i}\\ &\geq \overline{B}_k(p_i)\cdot\left(1-\frac{B_{k+1}}{1+(\beta-1)p_i}\right)-\frac{B_{k+1}p_i(\beta-1)\overline{B}_k(p_i)}{1+(\beta-1)p_i}\\ &= \overline{B}_k(p_i)(1-B_{k+1}) \;=\; \overline{B}_{k+1}(p_i).\end{aligned}$$

The inequality in the fourth line follows from the induction hypothesis and from $\overline{B}_k(1) \leq \overline{B}_k(p_i)$. □

Lemmas 3, 4, and 5 lead to an efficient deterministic approximation algorithm for the MMSC problem. First, the linear program $\mathrm{LP}_{\mathrm{MMSC}}$ has to be solved. The probabilites p_i are determined as described in the last section. For α and β as in Lemma 4, $P(p_1,\ldots,p_m)<4/5$. The probabilities p_i are now set to 0 or 1 such that $P(p_1,\ldots,p_m)$ remains smaller than $4/5$. This is possible by Lemma 5. When all $p_i \in \{0,1\}$, we have an integer solution for $\mathrm{IP}_{\mathrm{MMSC}}$. The probability that not all elements are covered or that the membership is larger than $\alpha\beta z'$ is smaller than $P < 4/5$. Because all p_i are 0 or 1, this probability must be 0. Hence, the computed $\mathrm{IP}_{\mathrm{MMSC}}$-solution is an $\alpha\beta$-approximation for MMSC:

Theorem 3. *For any MMSC instance, there exists a deterministic polynomial-time approximation algorithm with an approximation ratio of* $(1 + \mathrm{O}(1/\sqrt{z'}))(\ln(n)+1)$.

6 Conclusion

Interference reduction in cellular networks is studied in this paper by means of formalization with the *Minimum Membership Set Cover* problem. Although this combinatorial optimization problem appears to be a natural and simply describable problem

in the context of set covering, it has – to the best of our knowledge – not been studied before. We show using approximation-preserving reduction from the Minimum Set Cover problem that MMSC is not only NP-hard, but also that no polynomial-time algorithm can approximate the optimal solution more closely than up to a factor $\ln n$ unless $NP \subset TIME(n^{O(\log \log n)})$. In a second part of the paper this lower bound is shown to be asymptotically matched by an algorithm making use of linear programming relaxation techniques.

Finally, the question remains as an open problem, whether there exists a simpler greedy algorithm – considering interference increase during its execution – with the same approximation quality.

References

1. V. Chvátal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
2. G. Even, Z. Lotker, D. Ron, and S. Smorodinsky. Conflict-Free Colorings of Simple Geometric Regions with Applications to Frequency Assignment in Cellular Networks. In 43^{rd} *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.
3. U. Feige. A Threshold of ln n for Approximating Set Cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.
4. C. Fortuin, P. Kasteleyn, and J. Ginibre. Correlations Inequalities on Some Partially Ordered Sets. *Comm. Math. Phys.*, 22:89–103, 1971.
5. T. E. Harris. A Lower Bound for the Critical Probability in a Certain Percolation Process. *Proc. Cambridge Philos. Soc.*, 60:13–20, 1960.
6. J. Janssen. Channel Assignment and Graph Labeling. In I. Stojmenovic, editor, *Handbook of Wireless Networks and Mobile Computing*, chapter 5, pages 95–117. John Wiley & Sons, Inc., 2002.
7. D. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.
8. D. Kleitman. Families of Non-Disjoint Subsets. *Journal of Combinatorial Theory*, 1:153–155, 1966.
9. C.-J. Lu. A Deterministic Approximation Algorithm for a Minimax Integer Programming Problem. In *Proc. of the 10th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 663–668, 1999.
10. C. Lund and M. Yannakakis. On the Hardness of Approximating Minimization Problems. *Journal of the ACM*, 41(5):960–981, 1994.
11. L. Narayanan. Channel Assignment and Graph Multicoloring. In I. Stojmenovic, editor, *Handbook of Wireless Networks and Mobile Computing*, chapter 4, pages 71–94. John Wiley & Sons, Inc., 2002.
12. P. Raghavan and C. Thompson. Randomized Rounding: A Technique for Provably Good Algorithms and Algorithmic Proofs. *Combinatorica*, 7(4):365–374, 1987.
13. A. Srinivasan. Improved Approximations of Packing and Covering Problems. In *Proc. of the 27th ACM Symposium on Theory of Computing*, pages 268–276, 1995.

Routing and Coloring for Maximal Number of Trees*

Xujin Chen, Xiaodong Hu, and Tianping Shuai

Institute of Applied Mathematics
Chinese Academy of Sciences
P. O. Box 2734, Beijing 100080, China
{xchen,xdhu,shuaitp}@amss.ac.cn

Abstract. Let G be a undirected connected graph. Given g groups each being a subset of $V(G)$ and a number of colors, we consider how to find a subgroup of subsets such that there exists a tree interconnecting all vertices in each subset and all trees can be colored properly with given colors (no two trees sharing a common edge receive the same color); the objective is to maximize the number of subsets in the subgroup. This problem arises from the application of multicast communication in all optical networks. In this paper, we first obtain an explicit lower bound on the approximability of this problem and prove $\Omega(g^{1-\varepsilon})$-inapproximability even when G is a mesh. We then propose a simple greedy algorithm that achieves performance ratio $O(\sqrt{|E(G)|})$, which matches the theoretical bounds.

1 Introduction

All graphs considered in this paper are finite and undirected. Let G be a graph with vertex set $V(G)$ with $|V(G)| = n$ and edge set $E(G)$ with $|E(G)| = m$, and $\boldsymbol{\Gamma} = \{\Gamma_1, \ldots, \Gamma_g\}$ be a set of g groups, where each Γ_i is a subset of $V(G)$. A tree interconnecting Γ_i is a tree of G with $\Gamma_i \subseteq V(T_i)$. A family $\mathcal{T} = \{T_1, \ldots, T_g\}$ of trees is said to be a *tree family* of $\boldsymbol{\Gamma} = \{\Gamma_1, \ldots, \Gamma_g\}$ if there is a permutation ρ on $\{1, \ldots, g\}$ such that $T_{\rho(i)}$ is a tree interconnecting Γ_i for each $1 \leq i \leq g$. A *coloring* $\{(T_i, c_i) : i = 1, \ldots, t\}$ of a tree family $\{T_1, \ldots, T_t\}$ is called *proper* if tree T_i $(1 \leq i \leq t)$ receives color $c_i \in \{1, \ldots, c\}$ such that $c_i \neq c_j$ whenever $E(T_i) \cap E(T_j) \neq \emptyset$ for each i, where c is a positive integer and $\{1, \ldots, c\}$ is the set of available colors. In this paper we study how to find the maximum tree family of $\boldsymbol{\Gamma}$ that has a proper coloring. We call it *Maximum Tree Routing and Coloring* (Max-TRC) problem.

Our study on Max-TRC problem, besides its theoretical significance, is inspired by the application of multicast communications in all optical wavelength division multiplexing (WDM) networks. In a WDM network, nodes interested in some particular data make a *multicast group*, which requires a *multicast connection* for sending data from its source(s) to its destinations. Given a set of multicast connection requests, two steps are needed to set up the connections,

* Supported in part by the NSF of China under Grant No. 70221001 and 60373012.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 199–209, 2005.

routing and wavelength assignment. *Multicast routing* is to connect all members in each multicast group with a tree, so called *light-tree*. *Wavelength assignment* is to assign a wavelength to each of generated light-trees in such a way that no two trees sharing a common link are assigned the same wavelength. Since the number of wavelengths can be used in WDM networks is not large, how to make a good use of wavelengths becomes very important. This motivates extensive studies on the problem of multicast routing and wavelength assignment in WDM networks. A typical metric is *throughput*, the number of requests can be accepted (or satisfied) given a prespecified number of wavelengths, and the goal is to find optimal multicast routing and wavelength assignment for maximizing the throughput. Clearly, this is exactly MAX-TRC problem.

Lots of work has been done on the NP-hardness of some special versions of MAX-TRC problem (see Section 2), but the explicit approximability has not been obtained. The focus of this paper is to study the explicit inapproximability and approximability of MAX-TRC problem. The techniques introduced manipulate graph structures and enable us to obtain two major results (among others): (i) Unless $NP = ZPP$, MAX-TRC problem is not approximable within $g^{1-\varepsilon}$ for any $\varepsilon > 0$, even when the underlying topology is a mesh, and (ii) A simple greedy strategy for MAX-TRC problem has a nearly best possible approximation performance ratio of $O(\sqrt{m})$. To the best of our knowledge, this is the first time this problem is analyzed in terms of explicit inapproximability and approximability.

The remainder of this paper is organized as follows. In Section 2, we first introduce a few notations and then present some known results related to MAX-TRC problem. In Section 3, we prove the inapproximability results for trees, meshes and tori. In Section 4, we first propose a greedy algorithm for MAX-TRC problem in general graphs and prove its approximation performance ratio, and then we propose two approximation algorithms for two special classes of graphs. In section 5, we conclude this paper with some remarks.

2 Preliminaries

Given a graph G, the set of edges in $E(G)$ incident with a vertex $v \in V(G)$ is denoted by $\delta(v)$. An *independent set* in G is a set of pairwise nonadjacent vertices in $V(G)$. The MAXIMUM INDEPENDENT SET (MIS) problem is to find an independent set of largest cardinality $\alpha(G)$. MIS is a special case of the MAXIMUM k-COLORABLE INDUCED SUBGRAPH (MCIS) problem which is to find a maximum subset of $V(G)$ that is the union of k independent sets in G.

A tree family $\mathcal{T} = \{T_1, \ldots, T_g\}$ in graph G is usually associated with its *intersection graph* $G_{\mathcal{T}}$ with vertex set $V(G_{\mathcal{T}}) = \{v_1, \ldots, v_g\}$ and edge set $E(G_{\mathcal{T}}) = \{v_i v_j : T_i \text{ and } T_j \text{ share at least one edge in } G\}$. Clearly, an independent set S in $G_{\mathcal{T}}$ (resp. the union S of c independent sets in $G_{\mathcal{T}}$) corresponds to a set $\{T_i : v_i \in S\}$ of edge-disjoint trees in G (resp. a set $\{T_i : v_i \in S\}$ of trees in G that admits a coloring using colors in $\{1, \ldots, c\}$).

Most of previous works related to MAX-TRC problem focused on the special case where every group has only two members, thus in this case a tree intercon-

necting a group is simply a path connecting the two members. The MAX-TRC problem in this case is commonly known as *Maximum Path Routing and Coloring* (MAX-PRC) problem, and has been extensively studied for several topologies, such as trees, rings and meshes. MAX-PRC Problem is NP-hard for all these three topologies, and is approximable within 1.58 in trees [13], within 1.5 in rings [11], within $O(1)$ in 2-dimensional meshes [8, 13]. A simplified version of MAX-PRC Problem assumes that the set of paths is prespecified [12]. For this version it was also proved NP-hard in general (but polynomial-time solvable when the graph is a chain [12]) and inapproximable within m^{δ} for some $\delta > 0$ unless $NP = P$.

When the number of available colors reduces to one, MAX-TRC problem reduces to the *maximum edge-disjoint Steiner tree problem*, and its special case mentioned above is referred to as the *maximum edge-disjoint path* problem. The standard greedy approaches [13] guarantee that if the maximum edge-disjoint Steiner tree problem is approximable within r, then MAX-TRC problem is approximable within $1/(1-e^{-1/r})$. Nevertheless, even the maximum edge-disjoint path problem seems hard to approximate: the current-best approximation guarantee is $\sqrt{m}+1$ achieved through greedy selection of shortest paths [9].

Both the decision and optimization versions of MAX-TRC problem are closely related to MCIS problem and its special case, MIS problem. It was shown by Bellare, Goldreich, and Sudan [1] that MIS problem is inapproximable within $n^{\frac{1}{4}-\varepsilon}$ for any $\varepsilon > 0$ assuming $NP \neq P$. Since the faith in the hypothesis $NP \neq ZPP$ is almost as strong as $NP \neq P$, the following negative result from [7], as well as positive result from [6], explains the lack of progress on good approximation for MIS problem and MCIS problem.

Theorem 1. *(i) MIS problem is inapproximable within $n^{1-\varepsilon}$ for any $\varepsilon > 0$, unless $NP = ZPP$. (ii) MCIS problem is approximable within $O(n(\log\log n/\log n)^2)$.*

3 Inapproximability Analysis

In this section, we shall show that MAX-TRC problem is as hard as MIS problem. Then our inapproximability results follow from Theorem 1 one way or another. Roughly speaking, we assume the existence of an r-approximation algorithm A for MAX-TRC problem, and use A to design an r-approximation algorithm B for MIS problem.

3.1 Trees

When the underlying graph is a tree, the tree interconnection group Γ_i is uniquely determined by the group members in Γ_i. Our first inapproximable case occurs in the *star* graph which is a tree with at most one vertex (called *center*) of degree greater than one.

Theorem 2. *The* MAX-TRC *problem in trees is inapproximable within a ratio* $\max\{g^{1-\varepsilon}, m^{\frac{1}{2}-\varepsilon}\}$ *for any $\varepsilon > 0$, unless $NP = ZPP$.*

Proof. Suppose for a contradiction that for some $\varepsilon > 0$, there is a $\max\{g^{1-\varepsilon}, m^{\frac{1}{2}-\varepsilon}\}$-approximation algorithm A for the MAX-TRC problem in trees. Consider an arbitrary graph H with $V(H) = \{v_1, \ldots, v_n\}$ and $E(H) = \{e_1, \ldots, e_m\}$. We construct a star graph G with $m+1$ vertices and m edges by setting $V(G) := \{a, b_1, \ldots, b_m\}$ and $E(G) := \{ab_i : i = 1, \ldots, m\}$, where a is the center. Define $\boldsymbol{\Gamma} := \{\Gamma_1, \ldots, \Gamma_n\}$ by $\Gamma_i := \{a\} \cup \left(\bigcup_{e_j \in \delta(v_i)} b_j\right)$, $1 \le i \le n$. Let T_i be the unique tree in G interconnecting Γ_i for each $1 \le i \le n$. Then $\mathcal{T} = \{T_1, \ldots, T_n\}$ is the unique tree family of $\boldsymbol{\Gamma}$, and $G_{\mathcal{T}} = H$. Now algorithm B runs A on the instance $(G, \boldsymbol{\Gamma}, 1)$ and outputs $\{v_i : A \text{ outputs } T_i\}$. It is easy to see that B is an approximation algorithm for MIS problem and has performance ratio, $\max\{g^{1-\varepsilon}, m^{\frac{1}{2}-\varepsilon}\} = \max\{n^{1-\varepsilon}, m^{\frac{1}{2}-\varepsilon}\} = n^{1-\varepsilon}$, the same as that of A, a contradiction to Theorem 1. □

Notice that the star graph used in the above reduction has its center a very large degree. A natural question is: whether low degrees make MAX-TRC problem easier? The answer is almost negative as shown by our results on meshes and tori in the next subsection.

3.2 Meshes

This subsection is devoted to the proof of the following inapproximability result on 2-dimensional meshes and tori. (We will only prove the case of meshes since the proof for tori is similar.)

Theorem 3. *The* MAX-TRC *problem in meshes (tori) is inapproximable within* $\max\{g^{1-\varepsilon}, \frac{1}{3}m^{\frac{1}{4}-\varepsilon}\}$ *for any* $\varepsilon > 0$*, unless* $NP = ZPP$.

To prove the theorem, we first show a lemma. For graph H with $V(H) = \{v_1, \ldots, v_n\}$ and $E(H) = \{e_1, \ldots, e_m\}$, we define groups $\boldsymbol{\Gamma} = \{\Gamma_1, \ldots, \Gamma_n\}$ on a $5m \times 5m$ mesh G as follows. Assume the vertices in G are labelled as in the Cartesian plane with its corners located at $(0,0)$, $(0, 5m-1)$, $(5m-1, 0)$, and $(5m-1, 5m-1)$, respectively. Associate each edge e_j $(1 \le j \le m)$ in H with two vertex sets in G: $R_j := \{(\ell, 5j-k) : \ell = 0, 1, \ldots, 5m-1; k = 1, 2, 3, 4, 5\}$ and $S_j := \{(5j-k, \ell) : \ell = 0, 1, \ldots, 5m-1; k = 1, 2, 3, 4, 5\}$. Notice that R_j (resp. S_j) consists of vertices located on five consecutive rows (resp. columns) of G, and satisfies the following two properties

$$\text{Each of } \{R_1, \ldots, R_m\} \text{ and } \{S_1, \ldots, S_m\} \text{ is a (disjoint) partition of } V(G). \quad (1)$$

$$R_j \cap S_k \text{ induces a } 5 \times 5 \text{ submesh } G_{jk} \text{ of } G, \text{for } 1 \le j, k \le m. \quad (2)$$

Now corresponding to vertex v_i in G, the i-th group

$$\Gamma_i := \bigcup_{e_j \in \delta(v_i)} (R_j \cup S_j) \quad (3)$$

in G is defined as the union of $R_j \cup S_j$ for all e_j incident with v_i.

Lemma 1. *Let $\boldsymbol{\Gamma} = \{\Gamma_1, \ldots, \Gamma_n\}$ consist of n groups in $5m \times 5m$ mesh G as defined in (3), and let $\mathcal{T}' = \{T'_1, \ldots, T'_n\}$ be a tree family in G such that T_i is a tree interconnecting Γ_i, $1 \le i \le n$. Then*

(i) there is a tree family $\mathcal{T} = \{T_1, \ldots, T_n\}$ such that T_i is a tree interconnecting Γ_i, $1 \le i \le n$, and the intersection graph of $\mathcal{T}$ is H; and

(ii) there does not exist distinct $i, j, k \in \{1, \ldots, n\}$ such that $v_iv_j \in E(H)$ and T'_i, T'_j, T'_k are pairwise edge-disjoint in G.

Proof. To justify claim (i), let us first construct a tree family $\mathcal{T} = \{T_1, \ldots, T_n\}$ so that each T_i $(1 \le i \le n)$ is a tree obtained from its vertex set $V(T_i) := \Gamma_i$ by two steps. In the first step, for every e_j incident with v_i, we add five rows each connecting all vertices in $\{(\ell, 5j-k) : \ell = 0, 1, \ldots, 5m-1\}$, $1 \le k \le 5$. Then the horizontal edges on the five rows span R_j. Summing over all $e_j \in \delta(v_i)$, in total $5|\delta(v_i)|$ rows are added. In the second step, we use vertical edges with both ends in S_j for some $e_j \in \delta(v_i)$ to connect the $5|\delta(v_i)|$ rows and the rest vertices in Γ_i under the condition that the resulting graph is a tree. (Though there are many possible T_i's, it is not a hard task to pick any one of them.)

By the construction, it suffices to show that the intersection graph $G_{\mathcal{T}}$ of $\mathcal{T}$ is identical with H. Indeed, for every edge $v_hv_i = e_j$ in H, trees T_h and T_i in G share common edges on the rows that span R_j. On the other hand, for every pair of nonadjacent vertices v_h and v_i in H, since $\delta(v_h) \cap \delta(v_i) = \emptyset$, we deduce from (1) that $R_j \cap R_k \ne \emptyset \ne S_j \cap S_k$ for all $e_j \in \delta(v_h), e_k \in \delta(v_i)$. Therefore, combining the definitions of Γ_h and Γ_i (recall property (3)) and the constructions of T_h and T_i we see that T_h and T_i shares neither a common horizontal edge nor a common vertical edge. In other words, T_h and T_i are edge-disjoint. Thus $G_{\mathcal{T}} = H$ as desired.

We now prove claim (ii). Suppose on the contrary that $v_iv_j = e_p \in E(H)$ and T'_i, T'_j, T'_k are pairwise edge-disjoint. Since $e_p \in \delta(v_i) \cap \delta(v_j)$, by property (3), both T'_i and T'_j contain $(R_p \cup S_p) \subseteq \Gamma_i \cap \Gamma_j$. Take $e_q \in \delta(v_k)$. Obviously $e_p \ne e_q$. Recalling property (2), we have a 5×5 submesh G_{pq} in G induced by $R_p \cap S_q$. Note that the 25 vertices of G_{pq} are all contained in $\Gamma_i \cap \Gamma_j \cap \Gamma_k \subseteq V(T'_i) \cap V(T'_j) \cap V(T'_k)$, and hence every vertex in G_{pq} is incident with three distinct edges one from each of T'_i, T'_j, T'_k. Consequently none of T'_h, T'_i, T'_j can have a branching vertex in G_{pq}, and each of the 9 internal vertices of G_{pq} is a leaf of at least two of T'_i, T'_j, T'_k. Therefore there are in total at least 18 different paths in $T'_i \cup T'_j \cup T'_k$ connecting these leaves to the boundary of G_{pq} because every of T'_i, T'_j, T'_k has vertices outside G_{pq}. Two of those paths must have a common edge in G_{pq} as G_{pq} has only 16 boundary vertices. The two different paths are contained in exactly one tree in $\{T'_h, T'_i, T'_j\}$. It follows that this tree has a branching vertex in G_{pq}. The contradiction establishes claims (ii). □

Proof of Theorem 3. Suppose that for some $\varepsilon > 0$, there is a $\max\{g^{1-\varepsilon}, \frac{1}{3}m^{\frac{1}{4}-\varepsilon}\}$-approximation algorithm A for the MAX-TRC problem in 2-dimensional meshes. By Theorem 1, it suffices to present a polynomial time algorithm B which always finds an independent set of size at least $\alpha(H)/n^{1-\varepsilon}$ in any given graph H on n vertices. If H is a complete graph, then B outputs an arbitrary vertex of H. So

we assume that $\alpha(H) \geq 2$, and by examining all pairs of vertices in H, B can find two nonadjacent vertices v_1 and v_2 in H in square time,

Suppose $V(H) = \{v_1, \ldots, v_n\}$ and $E(H) = \{e_1, \ldots, e_{m'}\}$, respectively. Let groups $\boldsymbol{\Gamma} = \{\Gamma_1, \ldots, \Gamma_n\}$ on a $5m' \times 5m'$ mesh G be defined for H as in (3). Then the number of the edges in G is $m = 10m'(5m' - 1) < 50n^4$, and by Lemma 1(i) there is a tree family $\mathcal{T} = \{T_1, \ldots, T_n\}$ in G such that T_i interconnects Γ_i ($1 \leq i \leq n$) and $G_{\mathcal{T}} = H$. This implies that there is a subset $\mathcal{S}$ of $\mathcal{T}$ consisting of $\alpha(H)$ pairwise edge-disjoint trees. Observe that $\mathcal{S}$ is a solution to the instance $(G, \boldsymbol{\Gamma}, 1)$ of the Max-TRC problem with only one color available. Hence the optimal value $OPT(G, \boldsymbol{\Gamma}, 1) \geq |\mathcal{S}| = \alpha(H)$. Now running algorithm A on $(G, \boldsymbol{\Gamma}, 1)$, algorithm B yields a solution $\{T'_{i_1}, \ldots, T'_{i_\beta}\}$ consisting of β pairwise edge-disjoint trees interconnecting multicast groups $\Gamma_{i_1}, \ldots, \Gamma_{i_\beta}$. As a result, algorithm B outputs $S = \{v_1, v_2\}$ if $\beta = 2$ and $S = \{v_{i_1}, \ldots, v_{i_\beta}\}$ if $\beta \geq 3$. Notice that S is an independent set in G of size β (recall Lemma 1(ii)). Moreover, $\frac{\alpha(H)}{|S|} \leq \frac{OPT(G,\boldsymbol{\Gamma},1)}{\beta} = \frac{OPT(G,\boldsymbol{\Gamma},1)}{A(G,\boldsymbol{\Gamma},1)} \leq \max\left\{g^{1-\varepsilon}, \frac{1}{3}m^{\frac{1}{4}-\varepsilon}\right\} = \max\left\{n^{1-\varepsilon}, \frac{1}{3}m^{\frac{1}{4}-\varepsilon}\right\} = n^{1-\varepsilon}$ shows that $|S|$ approximates $\alpha(H)$ within $n^{1-\varepsilon}$. It follows that B is an $n^{1-\varepsilon}$-approximation algorithm for MIS problem. The proof is then finished. □

4 Approximability Results

In this section, we will first propose a simple greedy algorithm Greedy_Tree for Max-TRC problem in general graphs, and then two approximation algorithms for the Max-TRC problem in trees and rings.

4.1 Greedy Algorithm for General Graphs

The main philosophy of our greedy strategy is to produce trees of less edges whenever possible. This is based on a natural intuition: a tree of less edges potentially has more chances to use the same color with others, and therefore coloring more trees.

In order to carry out the greedy strategy, it is worth noting that finding a tree for each given group Γ with minimal number of edges is the classic *Minimum Steiner Tree* (MST) problem, which is NP-hard in general [2] and has a 2-approximation algorithm [10].

For a given instance $(G, \boldsymbol{\Gamma}, c)$ of Max-TRC problem, the implementation of algorithm Greedy_Tree consists of a number of iterations. In the $(i+1)$-th iteration, set $\boldsymbol{\Gamma}_i$ contains all currently unrooted multicast groups. For every $j = 1, \ldots, c$, let G_j^{i+1} be the subgraph of G obtained by removing all edges in the trees already colored with color j. Clearly, G_j^{i+1} contains a Steiner tree of Γ, for every $\Gamma \in \boldsymbol{\Gamma}_i$ whose connection can be established using color j. Subsequently, for every such Γ, compute a 2-approximate MST [10] of Γ in G_j^{i+1}; all these 2-approximations are put into a set $\mathcal{T}_i$ (Steps 5-7). When all js have been considered, every group in $\boldsymbol{\Gamma}_i$ whose connection can be established has at least a tree of $\mathcal{T}_i$, and every tree in $\mathcal{T}_i$ can be colored with an appropriate

color. If $\mathcal{T}_i = \emptyset$, then no more connection can be established and the algorithm terminates; else among all produced trees in $\mathcal{T}_i$, select the one with the minimum number of edges and color it with an appropriate color (Steps 9-10), and then proceed to the next iteration.

ALGORITHM GREEDY_TREE
Input A set $\boldsymbol{\Gamma}$ of g groups in graph G, and a set $\{1, \ldots, c\}$ of colors.
Output Routing and proper coloring $\mathcal{C} = \{(T_i, c_i) : i = 1, \ldots, t\}$.
// GREEDY_TREE$(G, \boldsymbol{\Gamma}, c) = t$
1. $i \leftarrow 0$, $\mathcal{C}_0 \leftarrow \emptyset$, $\boldsymbol{\Gamma}_0 \leftarrow \boldsymbol{\Gamma}$.
2. **while** $\boldsymbol{\Gamma}_i \neq \emptyset$ **do begin**
3. $\quad \mathcal{T}_i \leftarrow \emptyset$
4. $\quad$ **For** $1 \leq j \leq c$ **do**
5. $\quad\quad$ **while** G_j^{i+1} contains a tree interconnecting $\Gamma \in \boldsymbol{\Gamma}_i$ **do begin**
6. $\quad\quad \mathcal{T}_i \leftarrow \mathcal{T}_i \cup \{$ 2-approximate MST of Γ in $G_j^{i+1}\}$
7. $\quad\quad$ **end-while**
8. $\quad$ **If** $\mathcal{T}_i \neq \emptyset$ **then**
9. $\quad\quad$ Pick $T_{i+1} \in \mathcal{T}_i$ and $j \in \{1, ..., c\}$ s.t. $|E(T_{i+1})| = \min_{T \in \mathcal{T}_i} |E(T)|$ and $E(T_{i+1}) \cap \Big(\bigcup_{T:(T,j) \in \mathcal{C}_i} E(T) \Big) = \emptyset$
10. $\quad\quad \mathcal{C}_{i+1} \leftarrow \mathcal{C}_i \cup \{(T_{i+1}, j)\}$
11. $\quad\quad \boldsymbol{\Gamma}_{i+1} \leftarrow \boldsymbol{\Gamma}_i - \{\Gamma : T_{i+1}$ interconnects $\Gamma)\}$
12. $\quad$ **else** $\boldsymbol{\Gamma}_{i+1} \leftarrow \emptyset$
13. $\quad t \leftarrow i$, $i \leftarrow i + 1$
14. **end-while**
15. Output $\mathcal{C} \leftarrow \mathcal{C}_t$

Theorem 4. GREEDY_TREE *is a* $(\sqrt{2m}+1)$*-approximation algorithm for* MAX-TRC *problem.*

Proof. Clearly, GREEDY_TREE terminates after at most g iterations (Steps 2-14) since $|\boldsymbol{\Gamma}_{i+1}| \leq |\boldsymbol{\Gamma}_i| - 1$ for every i (see Steps 10-12). Additionally, Step 11 implies that at most one tree is output for one group. The correctness follows from Steps 4-11 which guarantee inductively that every $\mathcal{C}_i$ $(1 \leq i \leq t)$ is a solution to $(G, \boldsymbol{\Gamma}, c)$.

We now turn to estimate the performance ratio of GREEDY_TREE. It is obvious that $t \geq \min\{c, g\}$. If $c \geq g$, then the algorithm solves $(G, \boldsymbol{\Gamma}, c)$ optimally. So we assume $c < g$ and therefore $t \geq c$. Consider an optimal solution to $(G, \boldsymbol{\Gamma}, c)$ and let $\mathcal{S}$ consist of the trees of the optimal solution for the multicast groups unrouted by GREEDY_TREE. Therefore, we have

$$|OPT(G,\boldsymbol{\Gamma},c)| \leq |\mathcal{S}|+t, \text{and no } c+1 \text{ trees in } \mathcal{S} \text{ can share the same edge in } G. \quad (4)$$

If $\mathcal{S} = \emptyset$, then $t = |OPT(G, \boldsymbol{\Gamma}, c)|$ and we are done. So we assume $\mathcal{S} \neq \emptyset$ and consider an arbitrary $S \in \mathcal{S}$. Suppose S is a tree interconnecting group Γ. Observe in Step 11 Γ is contained in every of $\boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_t$. Since $\boldsymbol{\Gamma}_{t+1}$ must be empty (otherwise GREEDY_TREE should output at least $t + 1$ trees), by Step

5, we have $S \nsubseteq G\backslash\big(\bigcup_{T:(T,j)\in\mathcal{C}_t} E(T)\big)$ for every $1 \le j \le c$. On the other hand, it is clear that $\mathcal{C}_{c-1}$ does not use color k for some $k \in \{1,\dots,c\}$, so the first c rounds of **while** loop (Steps 2-14) always find $S \subseteq G\backslash\big(\bigcup_{T:(T,k)\in\mathcal{C}_{h-1}} E(T)\big) = G$ for $1 \le h \le c$. Hence we may take an integer $i(S) \ge c$ such that

$$S \subseteq \bigcup_{j=1}^{c}\Big(G \setminus \big(\bigcup_{T:(T,j)\in\mathcal{C}_{h-1}} E(T)\big)\Big) \text{ for each } 1 \le h \le i(S), \text{ and} \tag{5}$$

$$E(S) \cap \Big(\bigcup_{T:(T,j)\in\mathcal{C}_{i(S)}} E(T)\Big) \ne \emptyset \text{ for each } 1 \le j \le c. \tag{6}$$

Subsequently, for every tree $S \in \mathcal{S}$, we can find an $h(S) \le i(S)$ and *charge* S to a common edge $e^S_{h(S)} \in E(S) \cap E(T_{h(s)})$ of S and the tree $T_{h(S)}$ in a way that no two trees in $\mathcal{S}$ are both charged to the same edge of the same tree in $\{T_1,\dots,T_t\}$, i.e.,

$$\text{either } e^R_{h(R)} \ne e^S_{h(S)} \text{ or } h(R) \ne h(S), \text{ for any distinct } R, S \in \mathcal{S}. \tag{7}$$

To establish such a correspondence between trees in $\mathcal{S}$ and edges in $T_1,\dots,T_t$, we shall make use of a matching in a bipartite graph H as follows: the vertex set of H is disjoint union of independent set X and independent set Y for which $X := \mathcal{S}$ consists of $|\mathcal{S}|$ vertices one for a tree in $\mathcal{S}$, while $Y := \{e_i : e \in E(T_i), i = 1,\dots,t\}$ is considered a multiset of size $\sum_{i=1}^{t}|E(T_i)|$ so that every edge $e \in E(G)$ has a number of copies e_i in Y, each carrying an index i iff $e \in E(T_i)$. The edge-set of H contains an edge joining $S \in X$ and $e_h \in Y$ iff $h \le i(S)$ and $e \in E(S) \cap E(T_h)$ in G. Since, by Step 10, $\mathcal{C}_{i(S)}$ does not route any of $T_{i(S)+1},\dots,T_t$, it follows from (6) that in H every $S \in X$ has at least c neighbors in Y. For any $X' \subseteq X$, we use $N(X')$ to denote the set of vertices in $H \setminus X'$ each having a neighbor in X'. Clearly, $N(X') \subseteq Y$. If $|N(X')| < |X'|$ for some $X' \subseteq X$, then there exists $e_i \in N(X')$ which has at least $c+1$ neighbors in X', so these $c+1$ neighbors are $c+1$ trees in $\mathcal{S}$ sharing the same edge $e \in E(G)$, contradicting (4). Thus $|N(X')| \ge |X'|$ for every $X' \subseteq X$, and Hall's theorem [5] guarantees the existence of a matching in H that saturates every $S \in X$. Suppose e_h is the neighbor of S in this matching. We then define $h(S) := h$ and $e^S_{h(S)} := e$. From the structure of H, it is easy to see that (7) is satisfied.

Furthermore, since $h(S) \le i(S)$, by (5) and by Steps 5-7, $\mathcal{T}_{h(S)-1}$ contains a tree T interconnecting Γ with $|E(T)| \le 2|E(S)|$. In turn, from the choice of $T_{h(S)} \in \mathcal{T}_{h(S)-1}$ made in Step 9, we deduce that $|E(T_{h(S)})| \le |E(T)| \le 2|E(S)|$. Thus $|E(S)| \ge \frac{1}{2}|E(T_{h(S)})|$ for every $S \in \mathcal{S}$. Let $s_i := |\{S : S \in \mathcal{S}, h(S) = i\}|$ denote the number of trees in $\mathcal{S}$ that are charged to edges of T_i, $1 \le i \le t$, then $\sum_{i=1}^{t} s_i = |\mathcal{S}|$, and by (7), $s_i \le |E(T_i)|$. Recall from (4) that the total number of edges of all trees in $\mathcal{S}$ does not exceed cm. This yields $cm \ge \sum_{S\in\mathcal{S}}|E(S)| = \sum_{i=1}^{t}\sum_{S\in\mathcal{S},h(S)=i}|E(S)| \ge \sum_{i=1}^{t}\sum_{S\in\mathcal{S},h(S)=i}\frac{1}{2}|E(T_{h(S)})| = \frac{1}{2}\sum_{i=1}^{t}|E(T_i)|s_i \ge \frac{1}{2}\sum_{i=1}^{t} s_i^s$. Combining this with $t \ge c$, we have $|\mathcal{S}|/t = (\sum_{i=1}^{t} s_i)/t \le \sqrt{(\sum_{i=1}^{t} s_i^2)/t} \le \sqrt{2cm/t} \le \sqrt{2m}$. This, together with (4), yields the desired performance ratio $OPT(G,\boldsymbol{\Gamma},c)/t \le \sqrt{2m}+1$. The proof is then finished. □

One of our proof techniques borrows an idea used in the work [9] for *Disjoint Path Problem*. However, combining two approaches in [9] and [13] can only obtain a performance ratio $1/(1-e^{-1/(\sqrt{2m}+1)})$ for GREEDY_TREE, which is greater than $\sqrt{2m}+1$.

4.2 Approximation Algorithms for Special Graphs

When the underlying graph G is a tree, the tree family $\mathcal{T}$ for a given set $\boldsymbol{\Gamma}$ of groups is unique, and MAX-TRC problem is reduced to coloring as many trees in $\mathcal{T}$ as possible. Clearly, the algorithm proposed in [6] for MCIS problem on $G_{\mathcal{T}}$ carries over to the MAX-TRC problem on $\boldsymbol{\Gamma}$, and has an approximation ratio $O(g(\log\log g/\log g)^2)$.

When the size of a multicast group is upper bounded by a constant k [3], the maximum degree of any tree in $\mathcal{T}$ is no more than k. We call such a tree family a *k-tree family*. Notice that the MAX-TRC problem on k-tree family in trees is NP-hard even when $k=2$ [13].

Theorem 5. *The* MAX-TRC *problem in tree graphs is approximable within* $1/(1-e^{-1/k})$ *for any given k-tree family.*

To prove the theorem, we apply the idea of iterative application of an algorithm DISJOINT_TREES for finding a maximal set of edge-disjoint trees. The standard iterative method [13] goes as follows: First run DISJOINT_TREES on the tree family $\mathcal{T}$ to get a maximal set of edge-disjoint trees. All trees in this set are colored with color 1, and then removed from the current tree family. And then run DISJOINT_TREES on the remaining tree family to find the maximal set of edge-disjoint trees and color them using a new color. Repeat this process until either no color can be used or no more tree is left uncolored. It is shown in [13] that if DISJOINT_TREES is a k-approximation algorithm for finding a maximum set of edge-disjoint trees, then the iterative method provides a $1/(1-e^{-1/k})$-approximation for the MAX-TRC problem.

We now present a k-approximation algorithm DISJOINT_TREES. Let us root the tree G at an arbitrary vertex r. The *level* of a vertex $v \in V(G)$ is defined as the length of the path from r to v. We use ℓ to denote the highest level of vertices in G. Let T be a tree in G, the root of T is the vertex in T that has the lowest level, and the *level* of T is equal to the level of its root.

ALGORITHM DISJOINT_TREES
Input A tree family $\mathcal{T}$ in tree graph G.
Output A set $\mathcal{T}$ of edge-disjoint trees with $\mathcal{T} \subseteq \mathcal{T}$. // DISJOINT_TREES$(G,\mathcal{T}) = |\mathcal{T}|$
1. $i \leftarrow \ell - 1$, $\mathcal{T} \leftarrow \emptyset$
2. **While** $i \neq -1$ **do begin**
3. Find a maximal set $\mathcal{T}_i$ of edge-disjoint trees in $G \backslash \bigcup_{T\in\mathcal{T}} E(T)$ each of level i in G.
4. $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}_i$, $i \leftarrow i-1$
5. **end-while**
6. Output $\mathcal{T}$

Proof of Theorem 5. It suffices to show that there are at most k·DISJOINT_TREES $(G, \mathcal{T})$ edge-disjoint trees in any k-tree family $\mathcal{T}$. Note that $\mathcal{T}$ is the disjoint union of $\mathcal{T}_0, \ldots, \mathcal{T}_{\ell-1}$. Denote by $\mathcal{S}$ the subset of $\mathcal{T}$ consisting of a maximum number of edge-disjoint trees, and set $\mathcal{S}_j := \{S \in \mathcal{S} : S$ is edge-disjoint from every tree in $\cup_{i=j+1}^{\ell-1}\mathcal{T}_i$, and shares a common edge with some tree in $\mathcal{T}_j\}$, $\ell - 1 \geq j \geq 0$. Then the maximality in Step 3 implies $|\mathcal{S}| = \sum_{j=0}^{\ell-1} |\mathcal{S}_j|$. Since every tree in $\mathcal{S}_j$ is edge-disjoint from every tree in $\mathcal{T}$ of level higher than j, and shares a common edge with a tree in $\mathcal{T}$ of level j, every tree in $\mathcal{S}_j$ has a edge that is incident with a vertex of level j and contained in a tree in $\mathcal{T}_j$. It is easy to see that $|\mathcal{S}_j| \leq k|\mathcal{T}_j|$ for all $\ell - 1 \geq j \geq 0$. Thus we have $|\mathcal{S}| = \sum_{j=0}^{\ell-1} |\mathcal{S}_j| \leq k \sum_{j=0}^{\ell-1} |\mathcal{T}_j| = k|\mathcal{T}|$. □

When the underlying graph is a ring, a tree for a group is simply a path containing all vertices in the group. In this simple case, by combining techniques used in [4, 11] we can prove the following theorem.

Theorem 6. *The* MAX-TRC *problem in ring graphs has an 1.5-approximation algorithm.*

5 Conclusions

In this paper we have studied the hardness of approximation for routing and coloring maximum number of trees. The $\Omega(g^{1-\varepsilon})$-inapproximability established provides a lower bound for designing good approximation algorithms for MAX-TRC. In the positive aspect, we have shown that the general-purpose greedy strategy achieves general lower bound, while focusing on network topology and group size bring about improvements on approximating MAX-TRC in special networks.

As the future work, the counterpart of MAX-TRC - *Minimal Tree Routing and Coloring* (MIN-TRC) problem deserves research efforts, where the goal is to find a tree family $\mathcal{T}$ of the set of all given g groups and a coloring of $\mathcal{T}$ with minimum number of colors.

References

1. M. Bellare, O. Goldreich, and M. Sudan, Free bits and nonapproximability–towards tight results, *SIAM J. Comput.* **27** (1998), 804-915.
2. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, 1979.
3. J. Gu, X.-D. Hu, X.-H. Jia, and M.-H. Zhang, Routing algorithm for multicast under multi-tree model in optical networks, *Theoretical Computer Science* **314** (2004), 293-301.
4. U. I. Gupta, D. T. Lee, and Y.-T. Leung, Efficient algorithms for interval graphs and circular-arc graphs, *Networks* **12** (1982), 459-467.
5. P. Hall, On representatives of subsets, *J. London Math. Soc.* **10** (1935), 26-30.
6. M. M. Halldórsson, Approximations of weighted independent set and hereditary subset problems, *J. Graph Algorithms and Applications* **4**(1) (2000), 1-16.

7. J. Hastad, Clique is hard to approximate within $n^{1-\epsilon}$, *Acta Mathematica* **182** (1999), 105-142.
8. J. Kleinberg and E. Tardos, Disjoint paths in densely embedded graphs, *Proc. 36th Annual ACM Symp. Theory of Computing*, 1995, 52-61.
9. S. G. Kollipoulos and C. Stein, Approximating disjoint-path problems using greedy algorithms and packing integer programs, *Integer Programming and Combinatorial Optimization*, Houston, TX, 1998.
10. L. Kou, G. Markowsky, and L. Berman, A fast algorithm for steiner trees, *Acta Informatica* **15** (1981), 141-145.
11. C. Nomikos, A. Pagourtzis, and S. Zachos, Minimizing request blocking in all-optical rings, *IEEE INFOCOM* 2003.
12. C. Nomikos and S. Zachos, Coloring a maximum number of paths in graphs, *Workshop on Algorithmic Aspects of Communication*, Bologna, 1997.
13. P. J. Wan and L. Liu, Maximal throughput in wavelength-routed optical networks, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **46** (1998), 15-26.

Share the Multicast Payment Fairly

WeiZhao Wang[1], Xiang-Yang Li[1,*], and Zheng Sun[2,**]

[1] Illinois Institute of Technology, Chicago, IL, USA
wangwei4@iit.edu, xli@cs.iit.edu
[2] Hong Kong Baptist University, Hong Kong, China
sunz@comp.hkbu.edu.hk

Abstract. In this paper, we study how to share the payments to relay links among the receivers for multicast. Based on a strategyproof mechanism whose multicast tree is at most 2 times the optimal, we propose a payment sharing scheme that is $\frac{1}{n^2}$-budget-balanced, cross-monotonic, and in the core. We also prove that there is no payment sharing scheme that can achieve β-budget-balance and cross-monotonicity for $\beta = \Omega(\frac{1}{n})$. When both the relay agents and the receivers are selfish, we show a negative result: combining a strategyproof mechanism for the relay agents and a strategyproof sharing scheme for the receivers does not necessarily imply a strategyproof mechanism overall.

1 Introduction

Multicast has been a popular technique for supporting group-based applications, such as video-conference and content distribution. Multicast routing often uses a tree to connect the receivers to the source, and every internal node only sends the data to its downstream nodes in the tree, which saves the bandwidth. Traditionally, whenever the source needs to send some data to a subset of receivers, the multicast routing picks the shortest path tree (also called least cost path tree in [1]) that spans these receivers. This simple approach ignores the fact that the shortest path tree may be arbitrarily worse than the optimal tree with respect to the total cost. Thus, we need to find the optimal tree connecting a given set of receivers with the minimum total cost, a problem known as the *Steiner tree* problem. However, this problem is well-known to be NP-hard for both the node weighted and link weighted graph. Thus, a sequence of approximation algorithms have been developed for the Steiner tree problems. In this paper, we assume that the network is link weighted, *i.e.*, only the links have costs.

Recently, sharing the cost of the multicast routing among receivers in a "fair" manner has been studied extensively [2, 3]. An assumption made by the cost sharing schemes is that the costs of the links (or nodes) are publicly known. However, this is not the case in many application scenarios. When the links (or nodes) are independent and self-interested agents, several strategyproof mechanisms [1, 4, 5] have been developed so that a proper payment to each agent can

* The research of the author was supported in part by NSF under Grant CCR-0311174.
** The research of the author was supported in part by Grant FRG/03-04/II-21 and Grant RGC HKBU2107/04E.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 210–219, 2005.

be computed efficiently. When the payments to the agents needed to be shared among the receivers, we need to design a *payment sharing scheme* instead of the traditional cost sharing scheme. If every receiver is also selfish with a privately known valuation, then a *payment sharing mechanism* is needed to determine which receiver gets the multicast data and at what price. Surprisingly enough, several results from cost sharing mechanisms do not carry over to the payment sharing mechanism. In [1], Wang *et al.* first studied how to fairly share, among the set of receivers, the payment of the mechanism that uses the shortest path tree as its multicast tree. By assuming that each receiver is willing to pay the computed charge, *i.e.*, its valuation is sufficiently large, they proved that their payment sharing scheme is fair. However, the cost of the shortest path tree could be as large as r times of the optimum, where r is the number of receivers. In this paper, we will study the payment sharing scheme when the payment is computed by a certain mechanism using a multicast tree with a constant approximation ratio, when the network links are selfish agents with privately known costs. We first show that if a payment sharing scheme is cross-monotone and never overcharges, then the total charge to receivers is at most $\Theta(\frac{1}{n})$ of the total payment to the selfish relay links in the worst case. We then present a payment sharing scheme that is in the core and can recover at least $\Theta(\frac{1}{n^2})$ of the total payment to the selfish links. When both the relay agents and the receivers are selfish, we show a negative result: combining a strategyproof mechanism M for the relay agents and a fair sharing scheme ξ^{LST} for the receivers do not necessarily imply a strategyproof mechanism overall.

2 Preliminaries and Previous Works

2.1 Algorithmic Mechanism Design

In a non-cooperative game (such as multicast), there are n agents $\{1, 2, \cdots, n\}$. We assume each link is an agent for multicast. Each agent $i \in \{1, \cdots, n\}$ has some *private* information t_i, called its *type* (*e.g.* its cost to forward a packet in multicast). All agents' types define a *profile* $t = (t_1, \ldots, t_n)$. Each agent i declares a valid type τ_i' which may be different from its actual type t_i and all agents' strategies define a declared type vector τ. A mechanism $M = (\mathcal{O}, \mathcal{P})$ is composed of two parts: an output method $\mathcal{O}$ that maps a declared type vector τ to an output o, and a *payment* scheme $\mathcal{P}$ that decides the monetary payment $\mathcal{P}_i = \mathcal{P}_i(\tau)$ for every agent i. Each agent i has a valuation function $w_i(t_i, o)$ that expressed its preference over different outcomes. Agent i's *utility* is $u_i(t_i, o) = w_i(t_i, o) + p_i$, given output o and payment p_i. An agent i is said to be *rational* if it always chooses its strategy τ_i to maximize its utility u_i.

Let $\tau|^i t_i = (\tau_1, \cdots, \tau_{i-1}, t_i, \tau_{i+1}, \cdots, \tau_n)$. A mechanism $M = (\mathcal{O}, \mathcal{P})$ is *strategyproof* if it satisfies the following conditions. (1) **Incentive Compatibility (IC)**: $\forall$ agent $i, \forall \tau,\ w_i(t_i, \mathcal{O}(\tau|^i t_i)) + p_i(\tau|^i t_i) \geq w_i(t_i, \mathcal{O}(\tau)) + p_i(\tau)$. (2) **Individual Rationality (IR)**(a.k.a., Voluntary Participation): Each agent must have a non-negative utility. We always require that $\mathcal{O}$ and $\mathcal{P}$ are computed in polynomial time, which is called **Polynomial Time Computability (PC)**.

2.2 Payment Sharing

In multicast transmission, one of the major concerns is how to charge the receivers in a *fair* way. If the relay links are cooperative, *i.e.*, the costs of relay links are publicly known, we need to share the costs of the multicast tree among receivers fairly. For the fair cost sharing, most of the literatures [2, 6, 7] used the *Equal Link Split Downstream* (ELSD) sharing scheme to charge receivers: the *cost* of each link is shared *equally* among all its downstream receivers. If the relay links are selfish, we have to share the payments paid to these links, where the payment to a selfish link should be computed by a certain strategyproof mechanism. If we simply apply ELSD as our charging scheme, it usually is not fair in a common sense. Thus, we focus on the payment sharing in this paper.

Consider a set U of n players. For a subset $S \subseteq U$ of players, let $\mathbb{P}(S)$ be the total *payment* of providing service to S. A payment sharing scheme is simply a function $\xi_i(S, c)$ with $\xi_i(S, c) = 0$ for $i \notin S$, for every set $S \subseteq U$ of players. For payment sharing scheme, the definition of fairness is more subtle: many fairness concepts were proposed in the literature, such as *core* and *bargaining set* [8]. We call a charging scheme ξ *reasonable* or *fair* if it satisfies the following criteria.

1. **Budget Balance** (BB): The payment to all relay agents should be shared by the receivers, *i.e.*, $\mathbb{P}(R, c) = \sum_{i \in R} \xi_i(R, c)$. When budget-balance cannot be met, we relax it to β-budget-balanced: for the receiver set R, $\beta \cdot \mathbb{P}(R, c) \leq \sum_{i \in R} \xi_i(R, c) \leq \mathbb{P}(R, c)$, for some given parameter $\beta \leq 1$.
2. **No Positive Transfer** (NPT): Any receiver i's sharing should not be negative. In other words, we do not pay the receiver to receive service.
3. **Cross-monotonicity** (CM): For any two subsets $S \subseteq T$ and $i \in S$, $\xi_i(S, c) \geq \xi_i(T, c)$. In other words, the payment share of a receiver i should not go up if more receivers require the service. This is also called *population monotone*.
4. **Fairness under core** (Core): $\forall S \subseteq R$, $\sum_{i \in S} \xi_i(R, c) \leq \mathbb{P}(S, c)$.

Notice that a budget-balanced and cross-monotonic cost sharing scheme is always in the core. When each receiver q_i has a maximum payment ζ_i it is willing to pay to receive the multicast service, then we have to decide which receivers will get the service and at what price, *i.e.*, we need design a truthful mechanism. This mechanism sometimes is called *payment sharing mechanism*. A payment sharing mechanism satisfies *group strategyproof* if for any subset of receiver $S \subseteq R$, they can not collude together such that every receiver does not decrease its utility while at least one receiver increases its utility.

It is well-known [2] that a cross-monotonic budget-balanced *cost* sharing scheme ξ implies a group-strategyproof mechanism $M(\xi)$ that determines which receiver will get the service and at what price. Moulin and Shenker [2] also offered a characterization of a whole class of budget-balanced and group strategyproof mechanisms when the agents providing service is not selfish.

2.3 Problem Statement

Given a network $G = (V, E, c)$, where $V = \{v_1, \ldots, v_n\}$ is the set of terminals, $E = \{e_1, \ldots, e_m\}$ is the set of links. Every link e_i has a cost c_i to transmit a

unit size of data. Assume that each link is an individual agent who is selfish and rational. A set of receivers $R = \{q_1, \ldots, q_{|R|}\} \subset V$ are willing to receive the data from a source node s. For notational simplicity, we assume that $q_0 = s$ is the source node in one specific multicast and the size of the data is normalized to 1. To prevent monopoly, we assume that the graph G is bi-connected. We also assume that links will not *collude* to improve their profits together.

To design a multicast protocol, we first need to design a strategyproof mechanism $M^E = (\mathcal{O}^E, \mathcal{P}^E)$ such that the selected links form a topology (a tree, a mesh, a ring, etc) that spans the set of receivers R. This has been well-studied [1, 4, 5]. In this paper, we concentrate on designing a fair payment sharing scheme ξ when the payment is computed by a strategyproof mechanism $M^E = (\mathcal{O}^E, \mathcal{P}^E)$ whose multicast tree has a constant approximation ratio. We further assume that each receiver has a valuation of receiving the data from the source. Let ζ_i be the willing payment by receiver q_i, and ζ be the vector of the willing payments of the receivers. For simplicity of our notation, given a payment sharing scheme ξ, we always use $M(\xi)$ to denote its induced mechanism defined in [2]. If the links are not selfish, then this is the cost sharing problem, for which Jain and Vazirani [9] proposed a $\frac{1}{2}$-budget-balanced cross-monotonic cost sharing scheme in [9], which implies a $\frac{1}{2}$-budget-balanced, group-strategyproof cost-sharing mechanism. In this paper, we focus on designing a fair and strategyproof payment sharing scheme for multicast when relay links are selfish.

3 Payment Sharing for Multicast

3.1 Tree Construction and Payment Computation

In practice, the *shortest path tree* (SPT), which is the union of the shortest paths from source to all receivers, is most widely used as a multicast tree. We use $SPT(R, c)$ to denote the shortest path tree of a network when the network cost vector is c and receivers set is R. Notice the total cost of $SPT(R, c)$ could be as large as $|R|$ times the optimal tree. Takahashi and Matsuyama [10] gave a polynomial time algorithm computing a 2-approximation of the optimum tree. Then a series of results have been developed to improve the approximation ratio [11]. Due to its simplicity of construction, we will use algorithm in [10] to construct the multicast tree, and the resulting tree is denoted as $LST(R, c)$.

We now briefly review the truthful payment scheme for links when they are selfish. We continue to present some important properties stating the relations of different payment schemes, which are crucial to design our payment sharing scheme. Wang *et al.* [4] gave the truthful payment schemes for tree SPT and LST respectively. A more general framework to design the payment schemes for any given multicast structure is given in [1, 5]. We use $\mathcal{P}^{SPT}(R, c)$ and $\mathcal{P}^{LST}(R, c)$ to denote the payment scheme for tree SPT and LST respectively. We also use $\mathbb{P}^{SPT}(R, c)$ and $\mathbb{P}^{LST}(R, c)$ to denote the total payment to the links in the network based on the tree SPT and LST respectively.

Algorithm 1 Construct the Steiner Tree $LST(R, c)$ (see [10])

1: Initialize $LST(R, c) = \emptyset$.
2: **repeat**
3: **for** each receive qz_i in R **do**
4: Find the least cost path $\mathsf{LCP}(s, q_i, c)$ between s and q_i.
5: Find the receiver q_j with the minimum cost of the shortest path $\mathsf{LCP}(s, q_j, c)$.
6: $R \longleftarrow R - \{q_j\}$, add $\mathsf{LCP}(s, q_j, c)$ to $LST(R, c)$, and set all links' costs on $\mathsf{LCP}(s, q_j, c)$ as 0.
7: **until** R is empty.

For a link $e_k \in SPT(R, c)$, we compute an intermediate payment $p_k^i(c)$ to link e_k for any receiver q_i as $p_k^i(c) = |\mathsf{LCP}(s, q_i, c|^k\infty)| - |\mathsf{LCP}(s, q_i, c|^k 0)|$. The final payment to link $e_k \in SPT(R, c)$ is $\mathcal{P}_k^{SPT}(R, c) = \max_{q_i \in R} p_k^i(c)$.

For a link $e_k \in LST(R, c)$, the payment $\mathcal{P}_k^{LST}(R, c)$ is computed as follows:

Algorithm 2 Payment Scheme $\mathcal{P}^{LST}(R, c)$ for a link e_k on LST

1: Set $c_k = \infty$ and apply Algorithm 1. For simplicity, denote the cost vector in the beginning of ith round as $c^{(i)}$ and the path selected in round i as $\mathsf{P}(s, q_{\sigma_i})$.
2: **for** each round $i = 1, 2, \cdots, |R|$ **do**
3: Set $c_k^{(i)} = 0$.
4: Let $\mathsf{LCP}_{e_k}(s, q_t, c^{(i)})$ be the path with the smallest weight among all paths between s and receivers in R.
5: Define an intermediate payment $p_k^i(c)$ as $p_k^i(c) = |\mathsf{P}(s, q_{\sigma_i})| - |\mathsf{LCP}_{e_k}(s, q_t, c^{(i)})|$.
6: The final payment $\mathcal{P}_k^{LST}(R, c)$ is $\mathcal{P}_k^{LST}(R, c) = \max_{i=1}^{|R|} p_k^i(c)$.

Lemma 1. *Given a network $G = (V, E, c)$, $\mathcal{P}_k^{SPT}(R, c) \leq \mathbb{P}^{LST}(R, c)$.*

Proof. We prove it by contradiction. For the sake of contradiction, we assume that $\mathcal{P}_k^{SPT}(R, c) > \mathbb{P}^{LST}(R, c)$. Without loss of generality, we assume that $\mathcal{P}_k^{SPT}(R, c) = \mathbb{P}^{LST}(R, c) + \delta$ where $\delta > 0$. We also assume that $\mathcal{P}_k^{SPT}(R, c) = p_k^j(R, c)$, *i.e.*,$\mathcal{P}_k^{SPT}(R, c) = |\mathsf{LCP}(s, q_j, c|^k\infty)| - |\mathsf{LCP}(s, q_j, c|^k 0)|$, where $\mathsf{LCP}(s, q_j, c|^k 0)$ is the shortest path between s and q_j. Let $\hat{c} = c|^k(\mathcal{P}^{LST}(R, c) + \frac{\delta}{2})$. Notice here that $\mathcal{P}_k^{LST}(R, c)$ is the maximum cost that e_k could declare such that it is selected in tree LST (called cut-value in [5]). Thus, we have $e_k \notin LST(R, \hat{c})$. Let $\Pi^{LST}(s, q_j)$ be the path between s and q_j in the tree $LST(R, \hat{c})$, then $e_k \notin \Pi^{LST}(s, q_j)$. Notice that $\mathsf{LCP}(s, q_j, c|^k\infty)$ is the shortest path between node s and q_j when link e_k is removed. Thus

$$\mathcal{P}_k^{SPT}(R, c) \leq |\mathsf{LCP}(s, q_j, c|^k\infty)| \leq |\Pi^{LST}(s, q_j)| \leq |\mathsf{LCP}(s, q_j, \hat{c})|$$
$$\leq |\mathsf{LCP}_{e_k}(s, q_j, \hat{c})| \leq \mathbb{P}^{LST}(R, c) + \frac{\delta}{2} < \mathbb{P}^{LST}(R, c) + \delta = \mathcal{P}_k^{SPT}(R, c),$$

which is a contradiction. This finishes our proof. □

Similarly, we have the following lemma.

Lemma 2. *Given a network* $G = (V, E, c)$, $\mathcal{P}_k^{LST}(R, c) \leq \mathbb{P}^{SPT}(R, c)$.

Proof. Recall that if $e_k \notin LST(R, c)$, then $\mathcal{P}_k^{LST}(R, c) = 0$. Thus, we only need to consider the case when $e_k \in LST(R, c)$. Without loss of generality, we assume that $\mathcal{P}_k^{LST}(R, c) = p_k^i(c)$ (*i.e.*, the payment is maximized at the ith round) and $q_{\sigma_i} = q_j$. The tree shown in Figure 1 (a) is the tree at the beginning of iteration i, and the path $\Pi_3 = \mathsf{LCP}_{-e_k}(s, q_j, c)$. Now we discuss by cases:

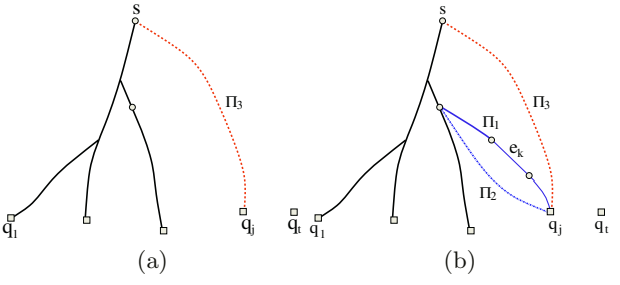


Fig. 1. The payment of LST. (a) the network at the beginning of ith iteration; (b) the network at the end of ith iteration

Case 1: $\mathsf{LCP}(s, q_j, c) = \mathsf{LCP}_{-e_k}(s, q_j, c)$, *i.e.*, link e_k is not on the shortest path between s and q_j. In this case, $|\Pi_3| = |\mathsf{LCP}(s, q_j, c)| \leq |SPT(R, c))| \leq \mathbb{P}^{LST}(R, c)$. Here $|H|$ denotes the total cost of links in H.

Case 2: $\mathsf{LCP}(s, q_j, c) = \mathsf{LCP}_{e_k}(s, q_j, c)$, *i.e.*, link e_k is on the shortest path between s and q_j. Recall that $\mathcal{P}_k^{SPT}(R, c) \geq |\mathsf{LCP}_{-e_k}(s, q_j, c)| - |\mathsf{LCP}(s, q_j, c|^k 0)|$. Thus, $|\mathsf{LCP}(s, q_j, c|^k 0)| = \sum_{e_i \in \mathsf{LCP}(s, q_j, c|^k 0) - \{e_k\}} c_i \leq \sum_{e_i \in SPT(R,c) - \{e_k\}} c_i \leq \sum_{e_i \in SPT(R,c) - \{e_k\}} \mathcal{P}_i^{SPT}(R, c)$. Therefore $|\Pi_3| = |\mathsf{LCP}_{-e_k}(s, q_j, c)| = \mathcal{P}_k^{SPT}(R, c) + |\mathsf{LCP}(s, q_j, c|^k 0)| \leq \mathcal{P}_k^{SPT}(R, c) + \sum_{e_i \neq e_k} \mathcal{P}_i^{SPT}(R, c) = \mathbb{P}^{SPT}(R, c)$.

This proves that $|\Pi_3| \leq \mathbb{P}^{SPT}(R, c)$ no matter whether e_k in on $\mathsf{LCP}(s, q_j, c)$ or not. Now we consider the ith iteration. For notational simplicity, we assume that $\Pi_1 = \mathsf{LCP}_{e_k}(s, q_t, c^{(i)})$ and $\Pi_2 = \mathsf{P}(s, q_{\sigma_i})$. From the assumption that $\mathcal{P}_k^{LST}(R, c) = p_k^i(c)$, $\mathcal{P}_k^{LST}(R, c) = p_k^i(c) = |\mathsf{P}(s, q_{\sigma_i})| - |\mathsf{LCP}_{e_k}(s, q_t, c^{(i)})| = |\Pi_2| - |\Pi_1| \leq |\Pi_3| - |\Pi_1| \leq |\Pi_3| \leq \mathbb{P}^{SPT}(R, c)$. This finishes our proof. □

Theorem 1. *For any graph* $G = (V, E, c)$, $\frac{\mathbb{P}^{LST}(R,c)}{n} \leq \mathbb{P}^{SPT}(R, c) \leq n \cdot \mathbb{P}^{LST}(R, c)$.

3.2 Payment Sharing Scheme

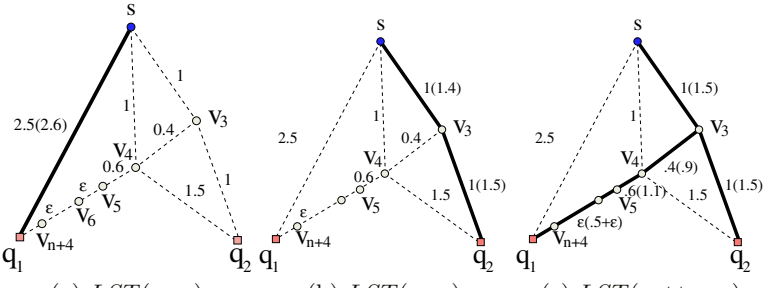
Recall that, when the links in the network are selfish, we should give links some payments that are least their costs. Thus, we need to share the payment instead of cost of the multicast tree among the receivers in a fair way, which is called *payment sharing*. For tree SPT, the ELSD scheme is not a fair payment sharing scheme [1]. A fair payment sharing scheme $\xi_i^{SPT}(\cdot)$ is also given in [1].

Theorem 2 (Wang, Li *et al.* [1]). *The payment sharing scheme $\xi_i^{SPT}(R,d)$ is fair, i.e., satisfies BB, NPT, NFR and CM.*

For a strategyproof mechanism based on LST, if a payment sharing scheme ξ is β-budget-balanced and cross-monotonic, then we have

Theorem 3. *If a payment sharing scheme ξ is β-budget-balanced and cross-monotonic, then $\beta = O(\frac{1}{n})$. Here n is the size of the network.*

Proof. We prove it by presenting a network example here. The network and the costs of the edges are shown in Figure 2. There are n nodes between v_4 and q_1. The cost of link $v_i v_{i+1}$ is ϵ, for $5 \le i \le n+3$. Let ξ^{LST} be a payment sharing



(a) $LST(q_1, c)$ (b) $LST(q_2, c)$ (c) $LST(q_1 \cup q_2, c)$

Fig. 2. A bad example of payment sharing of LST

scheme for the mechanism $(LST, \mathcal{P}^{LST})$ that is β-budget-balanced and cross-monotonic. Then from the β-budget-balance property we have $\xi_1^{LST}(q_1, c) \le \mathcal{P}_1(q_1, c) = 2.6$ and $\xi_2^{LST}(q_2, c) \le \mathcal{P}_1(q_2, c) = 2.9$. When the receiver set is $q_1 \cup q_2$, the cross-monotonicity property implies that $\xi_1^{LST}(q_1 \cup q_2, c) + \xi_2^{LST}(q_1 \cup q_2, c) \le \xi_1^{LST}(q_1, c) + \xi_2^{LST}(q_2, c) = 5.5$. Notice that $\mathcal{P}(q_1 \cup q_2) = 6 + 0.5 \cdot n$. Thus, $\beta \le \frac{\xi_1^{LST}(q_1 \cup q_2, c) + \xi_2^{LST}(q_1 \cup q_2, c)}{\mathcal{P}(q_1 \cup q_2)} = \frac{5.5}{6+0.5 \cdot n} = O(\frac{1}{n})$. This finishes our proof. □

The above theorem shows the limitations on the payment sharing scheme when the payment is computed by mechanism $(LST, \mathcal{P})$. In the following, we present a payment sharing scheme for LST that achieves $\frac{1}{n^2}$-budget-balance and cross-monotonicity, based on payment sharing scheme in [1].

Algorithm 3 Payment Sharing Scheme for LST

1: Compute the payment sharing $\xi_k^{SPT}(R, c)$ for each receiver q_k (see [1]).
2: For each receiver q_k, set $\xi_k^{LST}(R, c) = \frac{\xi_k^{SPT}(R,c)}{n}$, where n is the number of the nodes in G.

Theorem 4. *The payment sharing scheme defined in Algorithm 3 satisfies NPT, CM, $\frac{1}{n^2}$-budget-balance, and is in the core.*

Proof. Recall that $\xi_k^{LST}(R,c) = \frac{\xi_k^{SPT}(R,c)}{n}$. From Theorem 2, we obtain that $\xi^{LST}(R,c)$ satisfies NPT and CM directly. Thus, we only need to prove that $\xi^{LST}(R,c)$ is $\frac{1}{n^2}$-budget-balance and in the core.

To prove the $\frac{1}{n^2}$-budget-balance property, we need to show that $\frac{\mathbb{P}^{LST}(R,c)}{n^2} \leq \sum_i \xi_i^{LST}(R,c) \leq \mathbb{P}^{LST}(R,c)$. From Theorem 1, we have $\frac{\mathbb{P}^{LST}(R,c)}{n^2} \leq \frac{\mathbb{P}^{SPT}(R,c)}{n} \leq \mathbb{P}^{LST}(R,c)$. From Theorem 2, we know that $\xi^{SPT}(R,c)$ satisfies BB. Thus, $\frac{\mathbb{P}^{SPT}(R,c)}{n} = \frac{\sum_i \xi_i^{SPT}(R,c)}{n} = \sum_i \xi_i^{LST}(R,c)$. Consequently, we have $\frac{\mathbb{P}^{LST}(R,c)}{n^2} \leq \sum_i \xi_i^{LST}(R,c) \leq \mathbb{P}^{LST}(R,c)$. This proves that the payment sharing scheme is $\frac{1}{n^2}$-budget-balanced.

Since $\sum_i \xi_i^{LST}(R,c) = \sum_i \frac{\xi_i^{SPT}(R,c)}{n} = \frac{\mathbb{P}^{SPT}(R,c)}{n} \leq \mathbb{P}^{LST}(R,c)$, ξ^{LST} is in the core. This finishes our proof. ∎

Notice that there is a gap between the upper bound $O(\frac{1}{n})$ and lower bound $\Omega(\frac{1}{n^2})$ on β for β-budget-balanced cross-monotonic payment sharing scheme. A future work is to close the gap. We conjecture that $\Theta(\frac{1}{r \cdot n})$ is a bound for both.

3.3 Satisfy Budget Balance with γ-Relaxed Core

In Section 3.2, we present a payment sharing scheme $\xi^{LST}(R,c)$ that is $\frac{1}{n^2}$-budget balanced, cross-monotonic, and in the core. This payment sharing scheme will most likely run to deficit. However, under certain circumstances, one would like to achieve the budget balance while sacrifice some other properties such as core. Thus, we generalize the core property as follows to *γ-relaxed core*: for any receiver set $S \in R$, $\sum_{q_i \in S} \xi_i(R,c) \leq \gamma \cdot \mathcal{P}(S,c)$. Here γ is fixed. Following theorem shows that a payment sharing scheme that is α-budget balanced and core implies a payment sharing scheme that is budget balanced and $\frac{1}{\alpha}$-relaxed core.

Theorem 5. *For any payment sharing scheme ξ that is α-budget-balanced and core, then payment sharing scheme $\hat{\xi}_i(R,c) = \xi_i(R,c) \cdot \frac{\mathcal{P}(R,c)}{\sum_{q_i \in R} \xi_i(R,c)}$ is budget balanced and $\frac{1}{\alpha}$-relaxed core.*

4 Selfish Relay Links and Receivers

So far, we assume that the receivers will pay the fair amount of sharing of payment to receive data using multicast. However, in practice, each individual receiver often has a maximum valuation indicating how much it is willing to pay to participate the multicast. A receiver chooses to join if and only if the charge is at most its valuation. Furthermore, receiver could also be *non-cooperative* and selfish: it will always maximize its profit by manipulating its reported valuation, should it be possible. This makes the multicast design even harder when both the relay agents and the receivers could be selfish. It is well-known that a cost

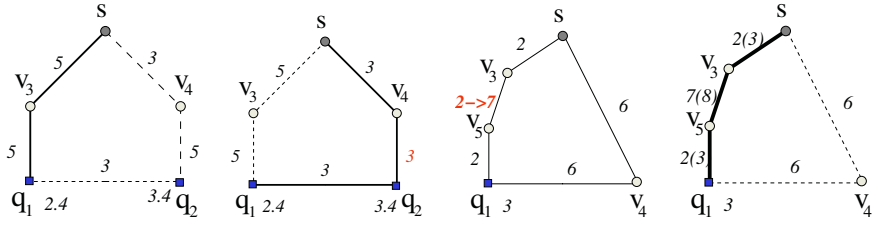
sharing scheme satisfying CM implies a *group-strategyproof* mechanism [2]. Thus, when each receiver q_i has a valuation ζ_i, the first intuition is that we can design a payment sharing mechanism as follows.

Algorithm 4 Payment Sharing Mechanism for Tree LST

1: $S \leftarrow R$, where R is the set of possible receivers.
2: **repeat**
3: Construct the tree $LST(S, c)$.
4: For each receiver $q_i \in S$, we compute the payment sharing $\xi_i^{LST}(S, c)$ based on the declared cost of all possible relay agents.
5: For each receiver $q_i \in S$, the receiver q_i is removed from S if $\xi_i^{LST}(S, c) > \zeta_i$, *i.e.*, $S \leftarrow S - \{q_i\}$ if $\xi_i(S, c) > \zeta_i$.
6: **until** no receiver is removed in this round
7: All remaining receivers S, denoted as $\hat{R} \subseteq R$, will receive the multicast data and pay a sharing $\xi_i^{LST}(\hat{R}, d) \leq \zeta_i$.

However, following theorem shows that the payment sharing mechanism defined by Algorithm 4 is not strategyproof.

Theorem 6. *Payment sharing mechanism defined by Algorithm 4 is not strategyproof, and moreover, some links may have incentives to lie up and lie down.*



(a) Original Tree (b) Tree after v_4q_2 lies (c) Original Tree (d) Tree after v_3q_5 lies

Fig. 3. A relay agent could either lie down or lie up its cost to improve its utility when use payment sharing mechanism 4

Figure 3 (a), (b) show that a link(v_4q_2) has incentive to lie down its cost from 5 to 3; and Figure 3 (c), (d) show that a link v_3v_5 has incentive to lie up its cost from 2 to 7. Examples where receivers may lie are omitted due to space limit. Theorem 6 shows that combining two strategyproof mechanisms for links and receivers does not imply a strategyproof mechanism overall.

5 Conclusion

Sharing the multicast cost among the receivers in a certain fair way has been studied widely in the literature. In this paper we studied how the payment

should be shared among the receivers when the payment is computed by a mechanism $M = (LST, \mathcal{P})$. Here, LST is the multicast tree construction method with approximation ratio 2. We described a payment sharing scheme that is $\frac{1}{n^2}$-budget-balanced, cross-monotonic, and in the core. We also proved that there is no payment sharing scheme that is β-budget-balanced and cross-monotonic for $\beta = \Omega(\frac{1}{n})$. When both the relay agents and the receivers are selfish, we showed a negative results: combining the strategyproof mechanism M and the fair payment sharing scheme $\xi^{LST}(\cdot)$ does not necessarily imply a strategyproof mechanism.

There are two future research directions. The first one is, for the β-budget-balanced and cross-monotonic payment sharing scheme for $M = (LST, \mathcal{P})$, to close the gap between the upper bound $\frac{1}{n}$ on β and the achievable lower bound $\frac{1}{n^2}$. The second direction is to design an overall strategyproof mechanism $M = (\mathcal{O}, \mathcal{P})$ that will form an approximately efficient multicast tree, decide the payment to each relay links, determine which receiver will receive the data and at what price. We have to make sure that both the relay links and receivers maximize their profit when they report their costs (or willing payment) truthfully.

References

1. Wang, W., Li, X.Y., Sun, Z., Wang, Y.: Design multicast protocols for non-cooperative networks. In: IEEE INFOCOM, 2005
2. Moulin, H., Shenker, S.: Strategyproof sharing of submodular costs: Budget balance versus efficiency. In: Economic Theory. Volume 18. (2001) 511–533
3. Feigenbaum, J., Papadimitriou, C.H., Shenker, S.: Sharing the cost of multicast transmissions. Journal of Computer and System Sciences **63** (2001) 21–41
4. Wang, W., Li, X.Y., Wang, Y.: Truthful multicast in selfish wireless networks. In: ACM Mobicom. 2004
5. Kao, M.Y., Li, X.Y., Wang, W.: Towards truthful mechanisms for binary selection problems: A general design framework. In: ACM EC. 2005
6. Feigenbaum, J., Papadimitriou, C.H., Shenker, S.: Sharing the cost of multicast transmissions. Journal of Computer and System Sciences **63** (2001) 21–41
7. Herzog, S., Shenker, S., Estrin, D.: Sharing the "cost" of multicast trees: an axiomatic analysis. IEEE/ACM Transactions on Networking **5** (1997) 847–860
8. Osborne, M.J., Rubinstein, A.: A course in game theory. The MIT Press (2002)
9. Jain, K., Vazirani, V.V.: Applications of approximation algorithms to cooperative games. In: ACM EC. (2001) 364–372
10. Takahashi, H., Matsuyama, A.: An approximate solution for the steiner problem in graphs. Mathematical Japonica **24** (1980) 573–577
11. Robins, G., Zelikovsky, A.: Improved steiner tree approximation in graphs. In: SIAM-ACM SODA. (2000) 770–779

On Packing and Coloring Hyperedges in a Cycle

Jianping Li[1,2,⋆], Kang Li[3], Ken C.K. Law[2], and Hao Zhao[2]

[1] Department of Mathematics, Yunnan University
Kunming 650091, P.R. China
jianping@ynu.edu.cn
[2] Department of Computer Science, City University of Hong Kong
Tat Chee Avenue, Kowloon, Hong Kong, P.R. China
{csjpli,cskckl}@cityu.edu.hk, zhaoh@cs.cityu.edu.hk
[3] School of Information Science and Engineering, Shandong University
Shandong 250100, P.R. China
kangli@sdu.edu.cn

Abstract. For a hypergraph and k different colors, we study the problem of packing and coloring some hyperedges of the hypergraph as paths in a cycle such that the total profit of the chosen hyperedges are maximized, here each link e_j on the cycle is used at most c_j times, each hyperedge h_i has a profit p_i and any two paths, each spanning all vertices of its corresponding hyperedge, must receive different colors if they share a link. This new problem arises in optical communication networks and it is called the *Maximum Profits of Packing and Coloring Hyperedges in a Cycle* problem (MPPCHC).

In this paper, we prove that the MPPCHC problem is *NP*-hard and present a 2-approximation algorithm. For the special case, where each hyperedge has the same profit and each capacity c_j is k, we propose a $\frac{3}{2}$-approximation algorithm to handle the problem.

Keywords: Minimum-cost flow, hyperedge, path coloring, approximation algorithm.

AMS Classifications: 90B10, 94C15.

1 Introduction

Ganley and Cohoon [4] proposed the *Minimum-Congestion Hypergraph Embedding in a Cycle* problem (MCHEC). The objective is to embed all hyperedges of a hypergraph as paths in a cycle such that the congestion, i.e., the maximum number of paths over any physical link of the cycle, is minimized, here the hypergraph has the same vertices as the cycle and a path spans all vertices of its corresponding hyperedge. This is a challenging problem with applications to various areas such as computer networks, multicast communication, parallel computation, electronic design automation.

Ganley and Cohoon [4] proved that the MCHEC problem is *NP*-hard for general hypergraphs, they provided a 3-approximation algorithm for the problem and gave an algorithm to determine whether the problem has an embedding

⋆ Correspondence author

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 220–229, 2005.

with congestion k or such an embedding does not exist in time $\mathcal{O}((nm)^{k+1})$, here n is the order of the cycle, a hypergraph has the same vertices as the cycle and m hyperedges. Gonzalez [6] derived two 2-approximation algorithms for the problem, Lee and Ho [8] developed a linear-time approximation algorithm to provide an embedding with congestion at most two times the optimum for the weighted version of the problem. There exist other approximation algorithms [2, 7] for the problem. Recently, Deng and Li [3] designed a polynomial time approximation scheme (PTAS) for the unweighted version of the problem.

In applications of the MCHEC problem, to set up a connection to all vertices in a hyperedge, a path is selected from many c-paths, each spanning all vertices of its corresponding hyperedge, and a wavelength is assigned to every link in the path. In all cases, when the wavelength assignments are made, there are no conflicts, i.e., no two connections (or paths) whose routes share a link can be assigned the same wavelength along that link. Current optical technologies impose limitations on the number of available wavelengths per fiber. This number is typically between 30 and 100.

When the number of wavelengths allowed in the network is fixed, it might be impossible to route the network to serve all hyperedges. In this case, one may try to route the network to serve as many hyperedges as possible. This reason leads us to redefine the problem as a variantly dual version of the MCHEC problem, called *Maximum Packing and Coloring Hyperedges in a Cycle* problem (MPCHC).

In this paper, we study the general version of the MPCHC problem, where each hyperedge has a profit, and the problem is to pack some hyperedges as paths in a cycle and to color these paths, the objective is to maximize the total profit of the chosen hyperedges, here each link e_j on the cycle is used at most c_j times and any two paths, each spanning the vertices of its corresponding hyperedge, must receive different wavelengths (i.e., colors) if they share a link. We call it as the *Maximum Profits of Packing and Coloring Hyperedges in a Cycle* problem (MPPCHC).

Let $\mathcal{P} = \{P_1, \ldots, P_q\}$ be a set of q paths on the cycle. The set $\mathcal{P}$ is *k-colorable* if we can assign k colors to the paths in $\mathcal{P}$ such that any two paths sharing a link on the cycle must receive different colors. For a subset $\mathcal{Q} \subseteq \mathcal{P}$ and each link e_j, the *load* $L(\mathcal{Q}, e_j)$ of e_j corresponding to $\mathcal{Q}$ is the number of paths in $\mathcal{Q}$ that contains the link e_j.

The MPPCHC problem is formally stated as follows:

INSTANCE: a set T of k colors, a cycle $G = (V, E)$ on n vertices, a hypergraph $G' = (V, H)$, a 'capacity' function $c : E \rightarrow \mathcal{Z}^+$ and a 'profit' function $p : H \rightarrow \mathcal{R}^+$.

QUESTION: Design a routing $\mathcal{Q}$ consisting of some paths on G, each path spanning all vertices of its corresponding hyperedge in H, such that $\mathcal{Q}$ is k-colorable, $L(\mathcal{Q}, e_j) \leq c_j$ holds for all $1 \leq j \leq n$ and $\sum_{P_i \in \mathcal{Q}} p_i$ is maximum.

There are two special cases for the MPPCHC problem: (1) The MPCHC problem is a special case of the MPPCHC problem, here each hyperedge has

the same profit and each capacity c_j is the number k of colors, and (2) The *Maximum Packing Hyperedges in a Cycle* problem (MPHC) is a special case of the MPPCHC problem, here the k-colorable constraint is omitted.

It is obvious that, for a hypergraph consisting of n vertices and m hyperedges, the MCHEC problem is solvable in polynomial time with the minimum congestion c if and only if the MPHC problem is solvable in polynomial time with the optimal value m, where each link capacity of the cycle is the same c. So the *NP*-hardness of the MCHEC problem [4] implies that the MPHC problem remains *NP*-hard, which shows that the MPPCHC problem is also *NP*-hard.

In this paper, we introduce some preliminaries and a fundamental algorithm in Section 2, then we present a 2-approximation algorithm for the MPPCHC problem in Section 3, and we also design a $\frac{3}{2}$-approximation algorithm for the MPCHC problem in Section 4. We conclude our work with some remarks and discussions on future work in the last section.

2 Preliminaries and Fundamental Algorithm

A cycle C consisting of n vertices is an undirected graph $G = (V, E)$ with vertex set $V = \{i | 1 \leq i \leq n\}$ and link set $E = \{e_i | 1 \leq i \leq n\}$, here each link e_i connects two vertices i and $i + 1$, $i = 1, 2, \ldots, n$, and we treat the vertex $n + 1$ as the vertex 1. Without loss of generality, we think the numbers on the vertices ordered in the clockwise direction. Let $H = (V, E_H)$ be a hypergraph with the same vertex set $V = \{i | 1 \leq i \leq n\}$ as C and the hyperedge set $E_H = \{h_1, h_2, \ldots, h_m\}$, here each hyperedge h_i is a subset of V with two or more vertices.

For each $1 \leq i \leq m$, a *connecting path* (simply a *c-path*) P_i in G for hyperedge h_i is a *minimal* path in G such that all vertices in h_i are in P_i, so the two end-vertices of P_i must in h_i. For a hyperedge h_i consisting of m_i vertices, i.e., $h_i = \{v_{j_1}, v_{j_2}, \ldots, v_{j_{m_i}}\}$, here v_{j_1}, v_{j_2}, …, $v_{j_{m_i}}$ are successively located on the cycle in the clockwise direction, there are exactly m_i possible c-paths for h_j, i.e., $P(v_{j_1}, v_{j_{m_i}})$, $P(v_{j_2}, v_{j_1})$, … and $P(v_{j_{m_i}}, v_{j_{m_i-1}})$, here $P(v_{j_t}, v_{j_{t-1}})$ $(1 \leq t \leq m_i)$ is the path from the vertex v_{j_t} to the vertex $v_{j_{t-1}}$ along the clockwise direction on G.

2.1 The MPC Problem

To design an approximation algorithm for the MPPCHC problem, our basic idea is to delete a suitable link $e_n = (n, 1)$ from the cycle G to get a chain L that consists of the same vertices as G. We need an optimal algorithm [9] on L for the MPC problem defined below, then we derive a 2-approximation algorithm to the MPPCHC problem.

The *Maximizing Profits in Chain Problem* (MPC) is stated as follows:

INSTANCE: a set T of k colors, a chain $L = (V, E)$ consisting of n vertices, a set of m paths $\mathcal{P} = \{P_1, \ldots, P_m\}$, a 'capacity' function $c : E \to \mathcal{Z}^+$ and a 'profit' function $p : \mathcal{P} \to \mathcal{R}^+$.

QUESTION: Find a subset $\mathcal{Q} \subseteq \mathcal{P}$ such that $\mathcal{Q}$ is k-colorable, $L(\mathcal{Q}, e_j) \leq c_j$ holds for all $1 \leq j < n$ and $\sum_{P_i \in \mathcal{Q}} p_i$ is maximized.

2.2 An Optimal Algorithm to the MPC Problem

We need an optimal algorithm to solve the MPC problem. The detailed algorithm can be found in Li, Li, Wang and Zhao [9], which is an extension of the Carlisle-Lloyd algorithm [1]. Here, we only outline the algorithm and the fundamental steps.

For $\mathcal{P} = \{P_1, \ldots, P_m\}$ in the instance $\mathcal{I}$ of the MPC problem on the chain L, where P_i has two end-vertices s_i and t_i ($s_i < t_i$) for $1 \leq i < m$, set $x_j = L(\mathcal{P}, e_j)$ for each link e_j ($1 \leq j < n$) and $p_{\max} = \max\{p_i : 1 \leq i \leq m\}$.

The new (directed) network is constructed as $N = (V, A; s, t; c, p)$, where V contains n vertices as L, the source s is the first vertex 1 and the sink t is the last vertex n, and A, c and p are defined in three cases:

Step 1. For $j = 1, 2, \ldots, n-1$, put a '*clique-arc*' $e_j = (j, j+1)$ into A with capacity k and cost *zero*, i.e., $c(e_j) = k$ and $p(e_j) = 0$;
Step 2. For $i = 1, 2, \ldots, m$, construct an '*interval-arc*' $e_i^* = (s_i, t_i)$ into A with capacity *one* and cost $-p_i$, i.e., $c(e_i^*) = 1$ and $p(e_i^*) = -p_i$;
Step 3. For $j = 1, 2, \ldots, n-1$, only if $\min\{k, x_j\} > c_j$, construct a '*dummy arc*' $e_j^{**} = (j, j+1)$ into A with capacity $\min\{k, x_j\} - c_j$ and cost $-p_{\max} m - 1$, i.e., $c(e_j^{**}) = \min\{k, x_j\} - c_j$ and $p(e_j^{**}) = -p_{\max} m - 1$.

The complete algorithm to the MPC Problem is given below:

Algorithm: Maximizing Profits in Chain (MPC)
INPUT: an instance $\mathcal{I}$ of the MPC problem.
OUTPUT: Find a subset $\mathcal{Q} \subseteq \mathcal{P}$ such that $\mathcal{Q}$ is k-colorable, $L(\mathcal{Q}, e_j) \leq c_j$ holds for all $1 \leq j < n$ and $\sum_{P_i \in \mathcal{Q}} p_i$ is maximized.
Step 1. Construct the directed network $N = (V, A; s, t; c, p)$;
Step 2. Compute the minimum-cost s-t flow for $N = (V, A; s, t; c, p)$;
Step 3. Construct the set $\mathcal{Q}$ of 'interval-arc's of value *one*, i.e., $\mathcal{Q} = \{e_i^* = (s_i, t_i) : f(e_i^*) = 1, 1 \leq i \leq m\}$;
Step 4. For all 'interval-arc's in $\mathcal{Q}$, assign the same color to the arcs that belong to the identical path sharing a unit of flow from the source s to the sink t. (There are at most k colors used since the flow f has its value k)
End of MPC

The MPC algorithm can solve the MPC problem optimally.

Theorem 1 (Li, Li, Wang and Zhao [9]) *Algorithm* MPC *solves the* MPC *problem in* $\mathcal{O}(k(2n + m))$ *time, where* n *is the number of vertices,* m *is the number of paths on the chain and* k *is the number of colors.* ∎

3 A 2-Approximation Algorithm for MPPCHC

The main difficulty for the MPPCHC problem is that there are $|h_i|$ choices to serve a hyperedge h_i in the cycle G. The basic ideas to solve the problem are:

(i) delete a suitable link e_j possessing the minimum capacity from G to obtain the chain L, (ii) use algorithm MPC in [9] for L to obtain an optimal solution to the MPC problem fixed on L, (iii) use the greedy algorithm to choose a set of $\min\{k, c_j\}$ suitable c-paths passing through the link e_j to have the heaviest profits, each c-path being corresponding to its corresponding hyperedge, and (iv) select the better solution obtained in (ii) and (iii). Hence, we give a 2-approximation algorithm.

Without loss of generality, we may assume that the link $e_n = (n, 1)$ has the minimum capacity on G, i.e., $c_n = \min\{c_j | 1 \le j \le n\}$, otherwise if the link $e_{j_0} = (j_0, j_0 + 1)$ has the minimum capacity on G, i.e., $c_{j_0} = \min\{c_j | 1 \le j \le n\}$, we can obtain a permutation

$$\sigma = \begin{pmatrix} 1 & 2 & \cdots & n - j_0 & n - j_0 + 1 & n - j_0 + 2 & \cdots & n \\ j_0 + 1 & j_0 + 2 & \cdots & n & 1 & 2 & \cdots & j_0 \end{pmatrix}$$

where all arithmetic involving integers are performed implicitly using modulo n operation if necessary, then we obtain that the link $e_{\sigma(n)} = (\sigma(n), \sigma(n) + 1)$ has the minimum capacity on G. We do the following operations (or steps) under the constraints where each integer j is replaced by the integer $\sigma(j)$, then we shall obtain the final results under the constraints where each integer j' is replaced by the integer $\sigma^{-1}(j')$, here σ^{-1} is the inverse permutation of σ, i.e.,

$$\sigma^{-1} = \begin{pmatrix} 1 & 2 & \cdots & j_0 & j_0 + 1 & j_0 + 2 & \cdots & n \\ n - j_0 + 1 & n - j_0 + 2 & \cdots & n & 1 & 2 & \cdots & n - j_0 \end{pmatrix}$$

We now assume that $e_n = (n, 1)$ has the minimum capacity on G. Let L be the chain obtained by deleting e_n from G. Then L has the same vertices as G. For each hyperedge h_i of a hypergraph, *different* from the facts that there exist $|h_i|$ c-paths on G to span all vertices in h_i, there exists a *unique* c-path on L to span all vertices in h_i, then we choose such a unique c-path on L to span all vertices in h_i (this c-path is also on G, but it does not pass through e_n in G).

For each hyperedge $h_i = \{v_{j_1}, v_{j_2}, \ldots, v_{j_{m_i}}\}$, we assume $1 \le v_{j_1} < v_{j_2} < \cdots < v_{j_{m_i}} \le n$. Since h_i determines a unique c-path $P_i = P(v_{j_1}, v_{j_{m_i}})$ on L to span all vertices in h_i such that P_i has two end-vertices v_{j_1} and $v_{j_{m_i}}$ on L, we may *exchangeably* use P_i and h_i to represent the same matter on L. For convenience, we assume that h_i has s_i and t_i ($s_i < t_i$) as two end-vertices of P_i on L and other internal vertices (if any) are omitted to be written in detail. We denote $h_i = \{s_i, \ldots, t_i\}$ ($s_i < t_i$).

Our algorithm for the MPPCHC problem is described as follows

Algorithm: Maximum Profits of Packing and Coloring Hyperedges in a Cycle (MPPC)

INPUT: an instance $\mathcal{I}$ of the MPPCHC problem.

OUTPUT: a feasible set of paths $\mathcal{Q}$, corresponding to the set of hyperedges H', such that $\sum_{P_j \in \mathcal{Q}} p_j$ is maximized.

Step 1. Choose a link $e_{j_0} = (j_0, j_0 + 1)$ to satisfy $c_{j_0} = \min\{c_j | 1 \le j \le n\}$, without loss of generality, we assume $j_0 = n$, i.e., $c_n = \min\{c_j | 1 \le j \le n\}$.

Step 2. Delete $e_n = (n, 1)$ from G, and then obtain the chain L.
Step 3. Put $D = \{\{s_i, t_i\} : h_j = \{s_i, \ldots, t_i\} \in H$ satisfies $s_i < \cdots < t_i$ for each $1 \le i \le m\}$.
Step 4. Use algorithm MPC on L with the pairs in D, the source $s = 1$ and the sink $t = n$. Let D' be the optimal set obtained (note $D' \subseteq D$).
Step 5. Put $H' = \{h_i = \{s_i, \ldots, t_i\} \in H : \{s_i, t_i\} \in D'$, where $s_i < \cdots < t_i\}$.
Step 6. Let $\mathcal{P}$ be the set of c-paths on L according to the pairs in D' (also corresponding to the hyperedges in H'), then each c-path in $\mathcal{P}$ has the two end-vertices s_i and t_i for some $\{s_i, t_i\} \in D'$ (also the hyperedge $h_j = \{s_i, \ldots, t_i\} \in H'$) and each $\{s_i, t_i\} \in D'$ uniquely determines a c-path in $\mathcal{P}$.
Step 7. Choose $\min\{k, c_n\}$ hyperedges in H to possess the heaviest profits, and then use $\mathcal{Q}$ to represents the set of such $\min\{k, c_n\}$ c-paths, each containing e_n to route, and H'' represents the set of such $\min\{k, c_n\}$ hyperedges in H.
Step 8. Output the better solution of $\mathcal{P}$ and $\mathcal{Q}$ corresponding its set of hyperedges (either H' or H'').
End of MPPC

Now, we have three facts: (i) since e_n has the minimum capacity on G, we choose $\min\{k, c_n\}$ hyperedges from H such that no link capacity is violated when we pack such $\min\{k, c_n\}$ hyperedges as paths in G, (ii) for each chosen hyperedge h_i consisting of $|h_i|$ vertices in step 7, we choose a path corresponding to its hyperedge from the $|h_i| - 1$ c-paths that pass through e_n to span all $|h_i|$ vertices in h_i, and (iii) we use the greedy algorithm (Schrijver [12], 2003) to assign k colors on all paths in $\mathcal{P}$, and obviously each of the $\min\{k, c_n\}$ paths in $\mathcal{Q}$ (corresponding to hyperedges), which pass through e_n, can be assigned to different color.

Theorem 2 *Algorithm* MPPC *is a* 2*-approximation algorithm for the* MPPCHC *problem and it runs in* $\mathcal{O}(k(2n + m))$ *time, where* n *is the number of vertices of the cycle* G*,* m *is the number of hyperedges of a hypergraph and* k *is the number of colors.*

Proof Let OPT be an optimal set of c-paths, as well as the optimal set of hyperedges, to the MPPCHC problem, and $OUT = \max\{\sum_{P_i \in \mathcal{P}} p_i, \sum_{P_j \in \mathcal{Q}} p_j\}$ the output value obtained from algorithm MPPC. Let OPT_1 be the set of c-paths in OPT that do not pass through e_n and OPT_2 the set of c-paths in OPT that pass through e_n. So OPT can be partitioned in two subsets OPT_1 and OPT_2, i.e., $OPT = OPT_1 \cup OPT_2$. For convenience, OPT, OPT_1 and OPT_2 are also as the total profits of all c-paths in OPT, OPT_1 and OPT_2, respectively.

From Theorem 1, we get $OPT_1 \le \sum_{P_i \in \mathcal{P}} p_i \le \max\{\sum_{P_i \in \mathcal{P}} p_i, \sum_{P_j \in \mathcal{Q}} p_j\}$. So we derive $OPT_1 \le OUT$.

We now show that the greedy algorithm (at step 7 from the algorithm) ensures $OPT_2 \le \sum_{h_j \in \mathcal{Q}} p_j$. Since e_n has the minimum capacity on G, the $\min\{k, c_n\}$ hyperedges chosen (at step 7 from the algorithm) satisfies the property that, when these $\min\{k, c_n\}$ hyperedges as paths are packed in G, no link capacity is violated. Let $\mathcal{Q} = \{P_{j_1}, \ldots, P_{j_{\min\{k,c_n\}}}\}$ and $OPT_2 = \{P_{j'_1}, \ldots, P_{j'_{m'}}\}$. Without loss of generality, we assume that $p_{j_1} \ge p_{j_2} \ge \cdots \ge p_{j_{\min\{k,c_n\}}}$ and

$p_{j'_1} \geq p_{j'_2} \geq \cdots \geq p_{j'_{m'}}$. Since OPT_2 is a *feasible* solution to an instance $\mathcal{I}$ of the MPPCHC problem, we get $\min\{k, c_n\} \geq m'$. By the fact that we choose the $\min\{k, c_n\}$ hyperedges corresponding to the c-paths possessing the heaviest profits at step 7, we obtain $p_{j'_l} \leq p_{j_l}$ for each $1 \leq l \leq m'$ ($\leq \min\{k, c_n\}$). Then $OPT_2 = \sum_{l=1}^{m'} p_{j'_l} \leq \sum_{l=1}^{m'} p_{j_l} \leq \sum_{l=1}^{\min\{k,c_n\}} p_{j_l} = \sum_{h_{j_l} \in \mathcal{Q}} p_{j_l} \leq \max\{\sum_{P_i \in \mathcal{P}} p_i, \sum_{P_j \in \mathcal{Q}} p_j\}$. So we get $OPT_2 \leq OUT$.

Hence, we conclude $OPT = OPT_1 + OPT_2 \leq 2OUT$, which implies that MPPC is a 2-approximation algorithm for the MPPCHC problem.

Now, we analyse the complexity of algorithm MPPC.

In step 1, it needs n steps to choose the link to have the minimum capacity from G. In steps 4, 5 and 6, Theorem 1 provides the complexity of $\mathcal{O}(k(2n+m))$. In step 7, it needs $\mathcal{O}(k)$ time to choose the $\min\{k, c_n\}$ hyperedges to possess the heaviest profits. Other steps need constant time to execute. Thus, algorithm MPPC runs in $\mathcal{O}(k(2n+m))$ time.

This completes the proof of Theorem 2. ∎

4 A $\frac{3}{2}$-Approximation Algorithm to MPCHC

Here, we study the MPCHC problem. Combining algorithm MPC and a technique of matching theory, we design a $\frac{3}{2}$-approximation algorithm for the MPCHC problem.

As in [10], two hyperedges $h_s = \{v_{i_1}, \ldots, v_{i_{m_s}}\}$ and $h_t = \{u_{j_1}, \ldots, u_{j_{m_t}}\}$ of a hypergeaph are *parallel* on the cycle G if there exist two c-paths P_{h_s} and P_{h_t} such that P_{h_s} and P_{h_t} contain no common link on G, here P_{h_s} spans all $|h_s|$ vertices in h_s and P_{h_t} spans all $|h_t|$ vertices in h_t. See Figure 1 (a). Otherwise, they are *crossing* on G, i.e., for each c-path P_{h_s} on G spanning all $|h_s|$ vertices in h_s and each c-path P_{h_t} on G spanning all $|h_t|$ vertices in h_t, P_{h_s} and P_{h_t} always contain at least one common link on G. See Figure 1 (b). If two hyperedges h_s and h_t are parallel on G, then there exist two c-paths P_{h_s} and P_{h_t} on G such that P_{h_s} and P_{h_t} are edge-disjoint on G, then they would be assigned by the same color. See Figure 1 (a). Before we design a $\frac{3}{2}$-approximation algorithm to the MPCHC problem, we construct an *auxiliary* graph $D = (H, E_H)$ corresponding to the set H of hyperedges of a hypergraph, where two hyperedges h_s and h_t of H is *adjacent* in D, i.e., $h_s h_t \in E_H$, if and only if they are parallel on G.

Now, we can design an algorithm to the MPCHC problem:

Algorithm: Maximum Packing and Coloring Hyperedges in a Cycle (MPCH)
INPUT: an instance $\mathcal{I}$ of the MPCHC problem.
OUTPUT: a feasible solution OUT such that $|OUT|$ is maximized.
Step 1 Delete the link $e_n = (n, 1)$ from the cycle G to obtain a chain L_1 with the source 1 and the sink n;
Step 2 Put $H_2 = \{\{s_i, t_i\} : \text{a hyperedge } h_j = \{s_i, \ldots, t_i\} \in H \text{ satisfies } s_i < \cdots < t_i \text{ for each } 1 \leq i \leq m\}$.
Step 3 Use algorithm MPC on L_1 with the pairs in H_2, the source $s = 1$ and the sink $t = n$. Let H'_2 be the optimal set obtained (note $H'_2 \subseteq H_2$).

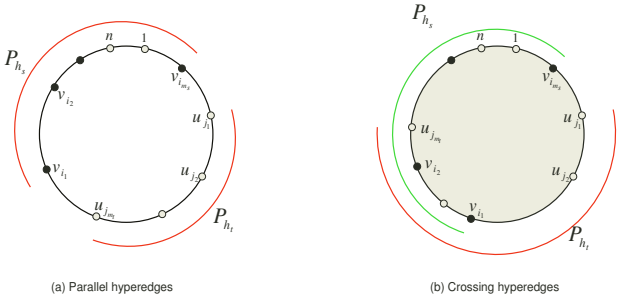


Fig. 1. Structures of two kinds of hyperedges

Step 4 Put $H' = \{h_i = \{s_i, \ldots, t_i\} \in H : \{s_i, t_i\} \in H_2'$, where $s_i < \cdots < t_i\}$.

Step 5 Let $\mathcal{P}$ be the set of c-paths on L_1 according to the pairs in H_2' (also corresponding to the hyperedges in H'), i.e., each c-path in $\mathcal{P}$ has the two end-vertices s_i and t_i for some $\{s_i, t_i\} \in H_2'$ and each $\{s_i, t_i\} \in H_2'$ uniquely determines a c-path in $\mathcal{P}$.

(*/ the set $\mathcal{P}$ is k-colorable and $L(\mathcal{P}, e_j) \le k$ holds for all $1 \le j < n$ /*)

Step 6 Construct the auxiliary graph $D = (H, E_H)$, and find a maximum (cardinality) matching M in D;

Step 7 Choose any $\min\{|M|, k\}$ edges from M, each corresponding to the two parallel hyperedges $\in H$; these $2 \times \min\{|M|, k\}$ parallel hyperedges, corresponding to the c-paths $\mathcal{Q}$, form an approximation solution;

(*/ the set $\mathcal{Q}$ is also k-colorable with exactly $2\min\{|M|, k\}$ paths /*)

Step 8 Output the better solution obtained in step 5 and step 7.

End of MPCH

Theorem 3 *Algorithm* MPCH *is a* $\frac{3}{2}$*-approximation algorithm to the* MPCHC *problem.*

Proof For an optimal solution OPT to the MPCHC problem and the approximation solution OUT by algorithm MPCH, let OPT_1 be the set of c-paths (corresponding to the hyperedges) in OPT that do not pass through e_n and OPT_2 to be the set of c-paths (corresponding to the hyperedges) in OPT that pass through e_n. So OPT can be partitioned in two subsets OPT_1 and OPT_2, i.e., $OPT = OPT_1 \cup OPT_2$. For convenience, OPT_2 also represents the set of colors that are assigned to the c-paths (corresponding to the hyperedges) in OPT_2.

Choose an optimal solution OPT such that the value $|OPT_2|$ is minimum among all optimal solutions to an instance $\mathcal{I}$ of the MPCHC problem. Then OPT_2 contains the least number of colors, i.e., there are minimum $|OPT_2|$ colors assigned to the c-paths in OPT_2, and each color in OPT_2 must be assigned to some c-paths in OPT_1. (Otherwise, if the color c in OPT_2 is not assigned to any c-path in OPT_1, we choose the c-path $P_{h_s}(v_{i_1}, v_{i_{m_s}})$ assigned by color c in OPT_2 corresponding to the hyperedge $h_s = \{v_{i_1}, \ldots, v_{i_r}, v_{i_{r+1}}, \ldots, v_{i_{m_s}}\}$, here $1 \le v_{i_{r+1}} < \cdots < v_{i_{m_s}} < \cdots < v_{i_1} < \cdots < v_{i_r} \le n$. See Figure 2 (a). Noting that

$P_{h_s}(v_{i1}, v_{i_{m_s}})$ passes through e_n along the clockwise direction to span all $|h_s|$ vertices in h_s from v_{i_1} to $v_{i_{m_s}}$, we construct a c-path $P_{h_s}(v_{i_r+1}, v_{i_r})$ to span all $|h_s|$ vertices in h_s from v_{i_r+1} to v_{i_r} along clockwise direction, then this new path does not pass through e_n and it is assigned the same color c. See Figure 2 (b). We obtain the other optimal solution such that the number of paths through e_n in this new optimal solution is exactly $|OPT_2| - 1$. This contradicts the selection of the optimal solution OPT.)

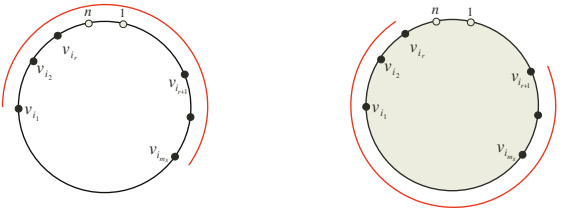


Fig. 2. Construction of P'_{h_s} from P_{h_s}

Since each color in OPT_2 must be assigned to some c-paths in OPT_1, we get the fact that each color in OPT_2 must color a pair of c-paths, one c-path in OPT_1 and the other in OPT_2. Then their corresponding hyperedges are adjacent in D The set of such $|OPT_2|$ pairs of hyperedges form a matching in D. By algorithm MPCH, M is a maximum (cardinality) matching in D, then we get $|OPT_2| \leq |M|$. Again, since each path in OPT_2 passes through e_n, we obtain $|OPT_2| \leq k$. Combining the above two facts, we get $|OPT_2| \leq \min\{|M|, k\}$.

On the other hand, Theorem 1 ensures that algorithm MPCH implies $|\mathcal{P}| \geq |OPT_1| = |OPT| - |OPT_2|$, then we obtain $|OPT| \leq |\mathcal{P}| + |OPT_2| \leq |\mathcal{P}| + \min\{|M|, k\}$.

By step 7 at algorithm MPCH, we have $2\min\{|M|, k\} = |\mathcal{Q}|$, i.e., $\min\{|M|, k\} = \frac{|\mathcal{Q}|}{2}$.

Hence, we conclude $|OPT| \leq |\mathcal{P}| + \min\{|M|, k\} = |\mathcal{P}| + \frac{|\mathcal{Q}|}{2} \leq \frac{3}{2}|OUT|$. This completes the proof of Theorem 3. ∎

5 Conclusion and Further Work

In this paper, we study the MPPCHC problem and its special version, then we derive a 2-approximation algorithm to the MPPCHC problem and a $\frac{3}{2}$-approximation algorithm to the MPCHC problem.

Since the MCHEC problem admits a PTAS by Deng and Li [3], we could consider to design a PTAS or a better approximation algorithms with factor less than $\frac{3}{2}$ to the MPCHC problem. We might ever find better approximation

algorithms for the MPPCHC problem with factor less than two. These provide some challenging and interesting problems to study in future.

Acknowledgements

Jianping Li is fully supported by the National Natural Science Foundation of China [Project No. 10271103], the Project-sponsored by SRF for ROCS, SEM, China, and Leadership in Teaching & Research of Department of Education of Yunnan Province.

Ken C K Law is partially supported by the City University Research Grant [Grant Ref: 7100255].

Hao Zhao are fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. Cityu 1196103E].

References

1. M.C. Carlisle and E.L. Lloyd, *On the k-coloring of intervals*, Discrete Applied Mathematics 59 (1995) 225-235. T.
2. T. Carpenter, S. Cosares, J.L. Ganley and I. Saniee, *A simple approximation algorithm for two problems in circuit design*, IEEE Trans. Compt. Vol. 47 (1998), no. 11, 1310-1312.
3. X.T. Deng and G.J. Li, *A PTAS for embeding hypergraph in a cycle* (extended abstract), accepted by the 31st International Colloquium on Automata, Languages and Programming (ICALP'04), July 12-16, 2004, Turku, Finland 2004.
4. J.L. Ganley and J.P. Cohoon, *Minimum-congestion hypergraph embedding in a cycle*, IEEE Trans. Comput. 46, No. 5 (1997) 600-602.
5. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, San Francisco (1979).
6. T.F. Gonzalez, *Improved approximation algorithms for embedding hyperedges in a cycle*, Information Processing Letter 67 (1998) 267-271.
7. Q. Gu and Y. Wang, *Efficient algorithm for embedding hypergraphs in a cycle*, in: Proceedings of the 10th International Conference on High Performance Computing, December 2003, Hyderabad, India.
8. S.L. Lee and H.J. Ho, *On minimizing the maximum congestion for weighted hypergraph embedding in a cycle*, Information Processing Letter 87 (2003) 271-275.
9. J.P. Li, K. Li, L.S. Wang and H. Zhao, *Maximizing Profits of Requests in WDM Networks*, to appear in Journal of Combinatorial Optimization, 2005.
10. C. Nomikos, A. Pagourtzis and S. Zachos, *Minimizing Request Blocking in All-Optical Rings*, Proceedings of IEEE INFOCOM (2003) 1355-1361.
11. P. Raghavan and E. Upfal, *Efficient routing in all-optical networks*, Proceeding of the 26th Annual ACM Symposium on the Theory of Computing (1994) 134-143.
12. A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, (Springer, The Netherlands, 2003).
13. R. Tarjan, Data Structures and Network Algorithms (SIAM, Philadelphia, PA, 1983).

Fault-Tolerant Relay Node Placement in Wireless Sensor Networks

Hai Liu, Peng-Jun Wan, and Xiaohua Jia

Department of Computer Science, City University of Hong Kong
Tat Chee Avenue, Kowloon, Hong Kong, P.R. China
{liuhai,pwan,jia}@cs.cityu.edu.hk

Abstract. The paper addresses the relay node placement problem in two-tiered wireless sensor networks. Given a set of sensor nodes in an Euclidean plane, our objective is to place minimum number of relay nodes to forward data packets from sensor nodes to the sink, such that: 1) the network is connected, 2) the network is 2-connected. For case one, we propose a $(6+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$ with polynomial running time when ε is fixed. For case two, we propose two approximation algorithms with $(24+\varepsilon)$ and $(6/T+12+\varepsilon)$, respectively, where T is the ratio of the number of relay nodes placed in case one to the number of sensors.

Keywords: sensor networks, fault-tolerant, relay node placement.

1 Introduction and Related Work

A sensor network is composed of a large number of sensor nodes that can be deployed on the ground, in the air, in vehicles, inside buildings or even on bodies. Sensor networks are widely deployed in environment monitoring, biomedical observation, surveillance, security and so on [4, 5]. Unlike the cellular networks and MANETs where there is unlimited energy supply in base stations or by batteries that can be replaced as needed, nodes in sensor networks have very limited energy supply and their batteries cannot usually be replaced due to special environments [6]. Since sensors' energy cannot support long range communication to reach a sink which is generally far away from the data source, multi-hop wireless connectivity is required to forward data to the remote sink. It is a key problem regarding how to gather data packets from sensor nodes to the sink in applications.

The two-tiered network architecture was proposed in [1, 2]. They employed relay nodes as gateways that are more powerful than sensor nodes in terms of energy storage, computing and communication capabilities. The network is partitioned into a set of clusters. The relay nodes act as cluster heads and they are connected with each other to perform the data forwarding task. (In the following, relay node, gateway and cluster-head refer to the same thing in the two-tiered sensor networks.) Each cluster has only one cluster-head and each sensor belongs to at least one cluster, such that sensor nodes can switch to

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 230–239, 2005.

backup cluster heads when current communication fails. In each cluster, sensor nodes collect raw data and report to the cluster-head. The cluster-head analyzes the raw data, extracts useful information, and then generates outgoing packets with much smaller total size to the sink through multi-hop path [3, 12]. Topology control in two-tiered wireless sensor networks has been discussed in [12]. The objective of the paper is to maximize the topological lifetime of the network, where the topological lifetime is defined as the lifetime of a network with regard to a given mission and the placement of sensor nodes, application nodes and base-stations. The authors proposed approaches to maximize the topological lifetime by arranging the base-stations location and the relay allocation. Similar to the works in [1, 2], two-tiered wireless sensor network architecture was proposed in [11] as a solution for structural health monitoring. The communication protocols used in the lower-tier (in clusters) and in the upper-tier (inter-clusters) were both described. Analysis in [11] showed that the maximum total number of sensor nodes that a network can handle is about 2000 4000 under current wireless data rates of 10Mbps.

Since a large number of nodes cooperate with each other in the network, fault-tolerance is one of important issues in wireless sensor networks. Communication faults in sensor networks can be caused by hardware damage, energy depletion, harsh environment conditions and malicious attacks. A fault in transmitter can cause the relay nodes to stop transmitting tasks to the sensors as well as relaying the data to the sink. Data sent by the sensors will be lost if the receiver of a relay node fails. So, a communication link failure to a sensor requires the sensor to be re-allocated to other cluster-head within communication range. If faults occur in inter-cluster-heads, the two corresponding cluster-head should be re-connected by another multi-hop path. That is, in order to handle general communication faults, there should be at least two node-disjoint paths between each pair of relay nodes in the network.

An intuitive objective of relay node placement is to place the minimum number of relay nodes to make the network connected, such that each sensor is covered by some relay nodes and all data packets can be gathered to the sink through these relay nodes (node a is covered by node b means that node a is within the communication range of node b). E. Biagioni et al. investigated placement problem of sensors in [8]. Three parameters were considered in the placement of sensors: resilience to single node failure, coverage of area of interest and minimizing the number of sensors. The authors showed that the choice of placement depends on sampling distance and communication radius. Different from the placement of sensor nodes, the placement problem of relay nodes in two-tiered sensor networks was discussed in [7]. The problem is to place the minimum number of relay nodes such that 1) each sensor node can communicate with at least two relay nodes and 2) the network of the relay nodes is 2-connected. An approximation algorithm was proposed and ratio was proved within $O(D \log n)$, where n is the number of sensor nodes in the network, and D the diameter of the network. Obviously, the proposed ratio is not a constant, which is a function of the size of input.

In this paper, we propose three approximation algorithms for the minimum relay-node placement problem (**MRP** in short). We discuss two cases of the MRP: 1) the network is connected and 2) the network is 2-connected. We proposed a $(6+\varepsilon)$-approximation algorithm for case one. We further proposed a $(24+\varepsilon)$-approximation algorithm and a $(6/T+12+\varepsilon)$-approximation algorithm for case two, respectively, for any $\varepsilon > 0$, where T is the ratio of the number of relay nodes placed to the number of sensors in case one.

2 System Model and Problem Specification

We first describe the MRP in two tiered wireless sensor networks, and then give a formal formulation of the problem.

Given a set of sensor nodes that are randomly distributed in a region and their location, in order to gather data packets from sensor nodes to the sink, we need to place some relay nodes to forward the data packets, such that each sensor node is covered by at least one relay node. Since sensor nodes have limited computing and communication capability, especially very limited energy resource, we assume that sensor nodes only report data packets to relay nodes within their communication range but not participate in data forwarding. That is, there is no direct link between any pair of sensor nodes. We further assume that all sensor nodes and all relay nodes have the same communication radius. The problem is to

1) place the minimum number of relay nodes in the region, such that the network (including sensor nodes and relay nodes) is connected;

2) place the minimum number of relay nodes in the region, such that there exists at least two node-disjoint paths between any pair of nodes (sensor nodes or relay nodes). That is, the whole network is 2-connected.

There are several major differences between our problem and the problem addressed in [7]. Firstly, the network of sensor nodes was assumed to be 2-connected in [7], we do not need this assumption. Secondly, sensor nodes also participate forwarding of data packets in [7] while they are not in our problem. Note that sensor nodes have very limited energy resource and computing capability. If they participate in data forwarding, it may cause early depletion of sensors, and make the network disconnected. Finally, placement of relay nodes in [7] only makes the network of relay nodes 2-connected, but we make the whole network 2-connected. Since our problem is more general than that in [7], our solution is also applicable to the problem in [7].

Now, we formally give the definition of the MRP problem.

Definition. Minimum Relay-node Placement problem (MRP): Given a set of sensor nodes S in a region and a uniform communication radius d, the problem is to place a set of relay nodes R, such that 1) the whole network G is connected and 2) G is 2-connected. The objective of the problem is to:

$$Minimize|R|$$

where $|R|$ denotes the number of relay nodes in R.

We first give a $(6+\varepsilon)$-approximation solution for the case one of MRP (MRP-1 in short), and then propose a $(24+\varepsilon)$-approximation and a $(6/T+12+\varepsilon)$-approximation solutions for MRP-2 by adding some relay nodes to make the network 2-connected.

3 Solution to MRP-1

Our solution is based on two foundational works. The first is the covering with disks problem. Given a set of points in the plane, the problem is to identify the minimum set of disks with prescribed radius to cover all the points. In [10], a polynomial time approximation scheme (PTAS) for this problem was proposed. That is, for any given error ε, the ratio of the solution found by the scheme to the optimal solution is not larger than $(1+\varepsilon)$. The running time is polynomial when ε is fixed. We call the scheme ***min-disk-cover scheme***. Different from covering with disks problem, MRP requires not only covering all sensor nodes, but also requires connection of the network, such that data packets can be gathered to the sink.

The other problem is the Steiner tree problem with minimum number of Steiner points (STP-MSP). Given a set of terminals in the Euclidean plane, the problem is to find a Steiner tree such that each edge in the tree has length at most d and the number of Steiner points is minimized. Du *et al.* proposed a 2.5-approximation algorithm for the STP-MSP [9]. We call the algorithm ***STP-MSP algorithm***. In our problem, sensor nodes do not relay messages for other nodes. So STP-MSP algorithm cannot be used to MRP problem directly.

Based on the min-disk-cover scheme and the STP-MSP algorithm, our solution for MRP-1 is composed in two steps. In step one, for any give error ε, we use min-disk-cover scheme to find a set of relay nodes that cover all sensor nodes. Note that the network of these relay nodes may not be connected if distance between them is larger than d. In order to connect these relay nodes, more relay nodes are needed. In step two, we run the STP-MSP algorithm and place additional relay nodes to get a Steiner tree. Finally, the Steiner tree and the sensor nodes form the connected network we desire. The algorithm is formally presented as follows.

$(6+\varepsilon)$-approximation algorithm for MRP-1

1. Place a set of relay nodes R_1 by using the min-disk-cover scheme, such that for $\forall s \in S, \exists r \in R_1$, and r cover s.
2. Place another set of relay nodes R_2 by running STP-MSP algorithm with input R_1.
3. Output $R_1 + R_2$.

Let R denote the set of all relay nodes we place in the above algorithm. Without loss of generality, we assume $R_1 \bigcup R_2 = \Phi$, i.e., $R = R_1 + R_2$ (or $R_1 \bigcup R_2$). Then the total number of relay nodes we place is $|R_1 + R_2|$. In the following theorem, we show that the algorithm for MRP-1 has ratio $(6+\varepsilon)$ for any $\varepsilon > 0$.

Theorem 1. Let R be our solution to MRP-1 and R^{opt} be the optimal solution to MRP-1. Then $\frac{|R|}{|R^{opt}|} \leq (6+\varepsilon)$.

Proof. Let R_1^{opt} denote the minimum set of relay nodes that cover S. Obviously, we have $|R_1^{opt}| \leq |R^{opt}|$. Since R_1 is the solution of PTAS to covering with disks problem, we have

$$|R_1| \leq (1+\varepsilon)|R_1^{opt}|. \tag{1}$$

Let R_2^{opt} denote the minimum set of relay nodes that makes R_1 connected. Since R_2 is the 2.5-approximation solution to STP-MSP problem, we have

$$|R_2| \leq 2.5|R_2^{opt}|. \tag{2}$$

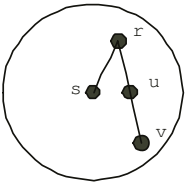


Fig. 1. Communication circle of sensor s.

In order to find out the relationship between R_2^{opt} and R^{opt}, we consider each relay node $r \in R_1$. For any $r \in R_1$, there must be at least one sensor node s which is covered by r (Otherwise, we can remove this useless r from R_1). We consider the communication circle of s (See Fig. 1). Note that for $\forall s \in S, \exists v \in R^{opt}$ such that both r and v can cover s. That is, relay nodes r and v are both in the communication circle of sensor s. So we have $d(r,v) \leq 2d$, where $d(r,v)$ is the Euclidean distance between r and v. If we place another relay node u in the middle point of edge (v,r), we have $d(u,v) = d(r,u) \leq d$. That is, node v can communicate with node r via node u.

For any relay node $r \in R_1$, we place another relay node according to the above description, such that R^{opt} can communicate with every relay node in R_1. That is, R^{opt} and these added relay nodes make R_1 connected. Note that R_2^{opt} is the minimum set of relay nodes that makes R_1 connected, and the number of added relay nodes is equal to $|R_1|$. So we have

$$|R_2^{opt}| \leq |R^{opt}| + |R_1|. \tag{3}$$

According to (1),(2),(3), the total number of relay nodes placed by the algorithm to MRP-1 is:

$$\begin{aligned} |R_1| + |R_2| &\leq (1+\varepsilon)|R_1^{opt}| + 2.5(|R^{opt}| + |R_1|) \\ &\leq (1+\varepsilon)|R^{opt}| + 2.5(|R^{opt}| + (1+\varepsilon)|R^{opt}|) \\ &\leq 6|R^{opt}| + 3.5\varepsilon|R^{opt}| \end{aligned}$$

Note that ε is an arbitrary positive number.
That is, $\frac{|R|}{|R^{opt}|} = \frac{|R_1|+|R_2|}{|R^{opt}|} \leq (6+\varepsilon)$

4 Solution to MRP-2

MRP-2 requires the network is 2-connected. It means that when one relay node fails, the network is still connected and data packet gathering operation can be carried out. Let T denote the ratio of the number of relay nodes place in MRP-1 to the number of sensors. That is, $T = |R|/|S|$. We first propose a $(24 + \varepsilon)$-approximation algorithm for general case of MRP-2, then improve the ratio to $(6/T + 12 + \varepsilon)$ if $T > 1/2$.

4.1 $(24 + \varepsilon)$-Approximation

Based on the solution R to MRP-1, our main idea is to add some backup nodes to the communication circle of each relay node r in R, such that the whole network is 2-connected. We first study the features these backup nodes should have. For$\forall r \in R$,

C_1. The backup nodes should cover all nodes in the communication circle of r. The first purpose of this condition is to make any node in the circle can switch to one of the backup nodes when r is out of service. The other purpose is to make at least one of the backup nodes communicate with outside when r is out of service. The first is for collecting messages purpose and the second is for forwarding messages purpose.

C_2. The backup nodes in the communication circle of r should communicate with each other. This is because that condition 1 only guarantees there exist at least one backup node can communicate with outside, but some other nodes in the circle may not communicate with outside if r is out of service. That is, the purpose of this condition is to make all nodes in the circle can communicate with outside.

We call the above two features ***2-connected conditions***. Are these two conditions sufficient? The following lemma shows that for each relay node r in R, if the backup nodes in the communication circle of r satisfy the 2-connected conditions, the network is 2-connected.

Lemma 1. Let $R^{'}$ denote the set of backup nodes in all communication circles of R. For $\forall r \in R$, if the backup nodes in the communication circle of r satisfy the 2-connected conditions, then the resulting network $S + R + R^{'}$ is 2-connected.

Proof. For $\forall v \in S + R + R^{'}$, we assume v fails. There are three cases: $v \in S, v \in R$ or $v \in R^{'}$. We prove the lemma case by case.

If $v \in S$, note that sensor nodes only report data packets to its cluster-head but not participate the data forwarding task, the network is connected.

If $v \in R^{'}$, since R is the solution to MRP-1 and the network $S + R$ is connected, there is no need to use backup nodes in this case.

If $v \in R$, according to the 2-connected conditions, all nodes in the communication circle of v are covered by backup nodes and can send their data packets to the outside. So the network is connected.

Since v is an arbitrary node in , Lemma 1 is proved.

According to Lemma 1, our algorithm is to add minimum number of backup nodes to the communication circle of each $r \in R$, such that the backup nodes

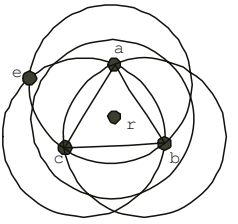


Fig. 2. Adding backup nodes to the communication circle of r.

satisfy the 2-connected conditions. For the condition C_1, obviously, adding two backup nodes can not cover the communication circle of r. That is, at least three backup nodes are needed in each communication circle. Based on R, our algorithm for MRP-2 is to add three backup nodes in the communication circle of each relay node in R, such that the network is 2-connected. The algorithm is formally presented as follows.

$(24+\varepsilon)$-approximation algorithm for MRP-2

1. Place a set of relay nodes R by running $(6+\varepsilon)$-approximation algorithm for MRP-1, such that $S+R$ is connected.
2. Place three backup nodes in the communication circle of each $r \in R$. The three backup nodes are placed on the three vertices of an equilateral triangle with length d. The center of the equilateral triangle is in r. See Fig. 2, where a, b, c are backup nodes. We denote the set of all backup nodes in this step by $R^{'}$.
3. Output $R + R^{'}$.

The following theorem claims the correctness of the algorithm.

Theorem 2. The set of backup nodes $R^{'}$ in the algorithm satisfy the 2-connected conditions and the final network $S + R + R^{'}$ is 2-connected.

Proof. In the following proof, we use *circlex* denote the communication circle of relay node x. We will prove the 2-connected conditions are satisfied one by one.

First, it is not difficult to see that the three new circles can cover the circle r. That is to say, any node in circle r is also in one of the three circles. See Fig. 2, since triangle abc is an equilateral triangle, we have

$$\angle ear = \angle eac + \angle car = 60^\circ + 30^\circ = 90^\circ$$
$$d(e,a) = d,$$
$$\text{then } d(e,r) > d.$$

It means that the circle a crosses the circle c on point e, which is outside the circle r. Likewise, points of intersection of circle r and circle b, circle c are both outside the circle r. So all nodes in the circle r can be covered by the union of circle a, circle b and circle c. The first condition C_1 is satisfied.

Second, since edge length of each equilateral triangle is d, which equals the communication radius of relay nodes. That means the three backup nodes can communicate with each other. It satisfies the condition C_2.

Associated with the Lemma 1, Theorem 2 is proved.

The following theorem states that the ratio of the above algorithm to MRP-2 is at most $(24+\varepsilon)$.

Theorem 3. Let $R+R^{'}$: our solution to MRP-2, R^{2-opt}: the optimal solution to MRP-2, we have $\frac{|R+R^{'}|}{|R^{2-opt}|} \leq (24+\varepsilon)$.

Proof. Note that for each communication circle of relay node in R, we add three backup nodes. We have $|R^{'}| = 3|R|$. So the total number of relay nodes we used to make the network 2-connected is:

$$\begin{aligned} |R+R^{'}| &= |R| + |R^{'}| = 4|R| \\ &\leq 4(6+\varepsilon)|R^{opt}| \\ &\leq 4(6+\varepsilon)|R^{2-opt}| \\ &= 24|R^{2-opt}| + 4\varepsilon|R^{2-opt}| \end{aligned}$$

Note that ε is an arbitrary positive number.

That is, $\frac{|R+R^{'}|}{|R^{2-opt}|} \leq (24+\varepsilon)$

In the next section, we will show that the ratio can be improved if $T > 1/2$.

4.2 $(6/T + 12 + \varepsilon)$-Approximation

After placing relay nodes by running the algorithm for MRP-1, the network is formed as a connected tree. Our main idea is to place backup nodes on each link of the tree to make the network 2-connected. The algorithm is formally presented as follows.

$(6/T + 12 + \varepsilon)$-approximation algorithm for MRP-2

1. Place a set of relay nodes R by running the $(6+\varepsilon)$-approximation algorithm for MRP-1, such that the tree $S+R$ is connected.
2. Place one backup node in the middle point of each link in the tree. We denote the set of all backup nodes in this step by $R^{''}$.
3. Output $R+R^{''}$.

The following theorem claims the correctness of the algorithm.

Theorem 4. The network $S+R+R^{''}$ is 2-connected.

Proof. To prove the theorem, we need to prove for $\forall v \in S+R+R^{''}$, if v fails, the resulting network is still connected. There are three cases: $v \in S, v \in R$ or $v \in R^{''}$. We prove the theorem case by case.

If $v \in S$, note that sensor nodes only report data packets to its cluster-head but not participate the data forwarding task, the network is still connected.

If $v \in R^{''}$, since R is the solution to MRP-1 and the network $S+R$ is connected, there is no need to use backup relay nodes in this case.

If $v \in R$, since v is a relay node, without loss of generality, we assume v forwards data packets from node u to node w (see Fig. 3, a, b are backup nodes). According to the above approximation algorithm, we have

$$d(a,v) = d(b,v) = 0.5d,$$

$$\text{then, } d(a,b) \leq d.$$

That is, the backup nodes a and b can communicate with each other. So the data packets can be forwarded to w via $u \to a \to b \to w$. The network is connected.

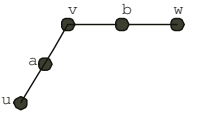


Fig. 3. Adding backup nodes a, b on the links of v.

Since v is an arbitrary node in $S + R + R^{''}$, Theorem 4 is proved.

The following theorem states that the ratio of the solution to MRP-2 can be improved to $(6/T + 12 + \varepsilon)$ if $T > 1/2$.

Theorem 5. Let $R + R^{''}$: our solution to MRP-2, R^{2-opt}: the optimal solution to MRP-2, we have $\frac{|R+R^{''}|}{|R^{2-opt}|} \leq (6/T + 12 + \varepsilon)$.

Proof. Since we place a node on each link in the tree which contains $|R + S|$ nodes in total, we have

$$|R^{''}| \leq |R + S| \leq |R| + |R|/T,$$

$$\begin{aligned} \text{then } |R + R^{''}| &\leq 2|R| + |R|/T \leq (2 + 1/T)(6 + \varepsilon)|R^{opt}| \\ &\leq (6/T + 12 + (2 + 1/T)\varepsilon)|R^{2-opt}|. \end{aligned}$$

Note that ε is an arbitrary positive number.

That is, $\frac{|R+R^{''}|}{|R^{2-opt}|} \leq (6/T + 12 + \varepsilon)$.

5 Conclusions

The fault-tolerant relay node placement problem in two-tiered wireless sensor networks was studied in the paper. Given a set of sensor nodes in a Euclidean plane, our objective is to place minimum number of relay nodes, such that 1) the network is connected and 2) the network is 2-connected.

Given an arbitrary positive number ε, we proposed a $(6 + \varepsilon)$-approximation algorithm for the problem one. The running time is polynomial when ε is fixed. We further proposed a $(24 + \varepsilon)$-approximation algorithm and a $(6/T + 12 + \varepsilon)$-approximation algorithm for problem two, where T is the ratio of the number of relay nodes placed in problem one to the number of sensors.

References

1. G. Gupta, M. Younis, "Fault-Tolerant clustering of wireless sensor networks",*Proceeding of IEEE WCNC'2003*, pp. 1579-1584.
2. G. Gupta, M. Younis, "Load-Balanced clustering of wireless sensor networks",*Proceeding of IEEE ICC'2003*, pp. 1848-1852.

3. J. Pan, Y. T. Hou, L. Cai, Y. Shi, S. X. Shen, "Topology control for wireless sensor networks",*Proceeding of ACM MOBICOM'2003*, pp. 286-299.
4. C.-Y. Chong and S. P. Kumar, "Sensor Networks: Evolution, Opportunities, and Challenges",*Proc. of the IEEE*, Vol 91, No. 8, Aug. 2003.
5. I. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, "A Survey on Sensor Networks",*IEEE Communications Magazine*, Aug. 2002.
6. P. Rentala, R. Musunuri, S. Gandham and U. Saxena, "Survey on Sensor Networks",http://citeseer.nj.nec.com/479874.html.
7. B. Hao, J. Tang and G. Xue, "Fault-tolerant relay node placement in wireless sensor networks: formulation and approximation", accepted by HPSR'2004.
8. E. Biagioni and G. Sasaki, "Wireless sensor placement for reliable and efficient data collection",*In Proceeding of the Hawaii International Conference on Systems Sciences*, Jan. 2003.
9. D. Du, L. Wang and B. Xu, "The Euclidean Bottleneck Steiner Tree and Steiner Tree with Minimum Number of Steiner Points", Computing and Combinatorics,*7th Annual International Conference COCOON 2001*, Guilin, China, Aug. 2001 Proceedings.
10. D. S. Hochbaum and W. Maass, "Approximation Schemes for Covering and Packing in Image Processing and VLSI",*Journal of the ACM (JACM)*, Vol 32, Issue 1, pp 130-136, Jan. 1985.
11. V. A. Kottapalli, A. S. Kiremidjian, J. P. Lynch, E. Carryer, T. W. Kenny, K. H. Law and Y. Lei, "Two Tiered Wireless Sensor Network Architecture for Structural Health Monitoring",*10th Annual International Symposium on Smart Structures and Materials*, San Diego, CA, USA, March 2-6, 2003.
12. Jianping Pan, Thomas Hou, Lin Cai, Yi Shi and Sherman X. Shen, "Topology Control for Wireless Sensor Networks", *MobiCom'03*, September 14-19, 2003, San Diego, California, USA.

Best Fitting Fixed-Length Substring Patterns for a Set of Strings*

Hirotaka Ono[1] and Yen Kaow Ng[2,**]

[1] Kyushu University, Department of Computer Science and Communication Engineering, 6-10-1, Hakozaki, Fukuoka, 812-8581, Japan
ono@csce.kyushu-u.ac.jp
[2] Kyushu Institute of Technology, Graduate School of Computer Science and Systems Engineering, Iizuka, 820, Japan
kalngyk@daisy.ai.kyutech.ac.jp

Abstract. Finding a pattern, or a set of patterns that best characterizes a set of strings is considered important in the context of Knowledge Discovery as applied in Molecular Biology. Our main objective is to address the problem of "over-generalization", which is the phenomenon that a characterization is so general that it potentially includes many incorrect examples. To overcome this we formally define a criteria for a most fitting language for a set of strings, via a natural notion of *density*. We show how the problem can be solved by solving the membership problem and counting problem, and we study the runtime complexities of the problem with respect to three solution spaces derived from unions of the languages generated from fixed-length substring patterns. Two of these we show to be solvable in time polynomial to the input size. In the third case, however, the problem turns out to be NP-complete.

1 Introduction

Extracting a common description for a set of strings is a central objective in many fields of research such as Knowledge Discovery and Artificial Intelligence. This description is sometimes expressed in a form of languages generated by some patterns, such as the one introduced by Angluin [1]. In the context of such patterns, the problem has received very intensive treatment in the field of Molecular Biology, where the patterns discovered are commonly called *consensus patterns* [4, 6].

A common problem in finding a language that best characterizes a set of examples is that of avoiding "over-generalization". A language that includes all the correct examples is called a *generalization* of the examples. It is called an *over-generalization* when it is overly general so that many potentially incorrect examples are also included. It is in general difficult to avoid over-generalization

* This work was partially supported by the Scientific Grant in Aid of the Ministry of Education, Science, Sports, Culture and Technology of Japan.
** The second author is supported by the the Japanese Government Scholarship of the Ministry of Education, Science, Sports, Culture and Technology of Japan.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 240–250, 2005.

in the absence of incorrect examples. To overcome this Angluin [1] and Shinohara [11] considered finding languages which are in a sense "minimal". More precisely, they considered finding a pattern language L covering a set of strings, such that no other pattern language that covers the strings is a proper subset of L. The solution space was extended to a subclass of the (bounded) unions of pattern languages by Arimura *et al.* [3, 10, 12], using a polynomial time algorithm called k-MMG. However, no algorithm has yet been proposed for the general case, and the output of k-MMG is dependent on the search path traversed [2]. Also, we note that there is another notion of "minimality" not guaranteed using their approach. We illustrate this with an example: let $S = \{$"abc", "bca"$\}$, and consider the language generated by the pattern "$*a*$" ("$*$" can be replaced with any string). While it does not contain any smaller proper subset which also contain S, it is arguably an inferior characterization compared to "$*bc*$".

We suggest a strategy to overcome this shortcoming. Our idea is to exclude as many as possible of all the strings which has not appeared in the input strings – up to a certain length. More precisely, given any input set S of strings, we consider patterns where their languages include S, while excluding as many as possible of the strings outside of S that are not longer than the longest string in S. In this sense the solution we aim at is "minimal" with respect to the set of all the strings of length up to that of the longest string seen. We define a notion of "density" which naturally expresses this idea, and ask the following questions: (1) with respect to a language and a set of examples, how efficiently can this density measure be computed, and (2) with respect to a set of examples, are there efficient computations for finding a language with the highest density? In this paper we study this measure as applied to a subclass of the pattern languages, namely the *substring patterns*.

This paper is structured as follows. We first introduce our notion of "density" and formally define the problem of finding the most fitting pattern languages for a set of strings. We then show how such densities can be computed in polynomial-time for the languages generated by the substring patterns. Finally, we study the hardness of finding a language of substring patterns that maximizes this density.

2 Preliminaries

Let N and N^+ denote the set of natural numbers and the set of positive integers, respectively. Cardinality of a set S is denoted by $|S|$. max(.), min(.) denote maximum and minimum of a set, where by convention $\max(\emptyset) = 0$ and $\min(\emptyset) = \infty$. A *word* over a non-empty alphabet A is a finite string of symbols taken from A. The *empty word* is a null string. The symbols A^+, $A^{=m}$ and $A^{\leqq m}$ denote the sets of non-empty words, words of length m, and words of length m or less, respectively. Let $A \setminus B$ denote the set $\{x \mid x \in A \text{ and } x \notin B\}$.

Let Σ be a fixed finite alphabet with at least 2 elements, and $V = \{x_1, x_2\}$ be an alphabet disjoint from Σ. Elements in Σ are called *constants* and elements in V are called *variables*. Each constant and variable has unit length. A non-empty word over Σ is called a *constant word*, or simply a *string*. A non-empty word

over $V \cup \Pi$ is called a *pattern*. The concatenation of two patterns p_1 and p_2, with p_2 as suffix, is written $p_1 p_2$. An arbitrary string of length m is written "$?^m$", and "$?^1$" is written simply "?". The length of a pattern p is denoted by $|p|$. For a set of patterns P, the length of the longest word in it is denoted by $\lceil P \rceil$, the length of the shortest word is denote by $\lfloor P \rfloor$, while the sum of all the lengths is denoted by $\| P \|$.

Let $\boldsymbol{P}$ denote the set of all patterns. A pattern is *regular* if each variable in it appears at most once. Let $\boldsymbol{RP}$ denote the set of all regular patterns. A pattern is a *substring pattern* just in case it is of the form "$x_1 s x_2$" and s is a non-empty word over Σ. Let $\boldsymbol{Psub}$ denote the set of all substring patterns. We also denote the set of all substring patterns "$x_1 s x_2$" where $|s|$ is of length l by $\boldsymbol{Psub}_l$. For each $l \in N^+$, we let $\boldsymbol{Psub}_l^* = \{P \mid P \subseteq \boldsymbol{Psub}_l\}$. For a set T of strings, $P(T)$ denotes a set of substring patterns constructed from T, i.e., $P(T) = \{x_1 s x_2 \mid s \in T\}$. Two strings t_1 and t_2 are said to *overlap* if either some suffix of t_1 matches some prefix of t_2, or some suffix of t_2 matches some prefix of t_1. For a set T of strings, we say T is *non-overlapping* if no string in T overlaps with itself (except when fully overlapping), and any two strings in T are non-overlapping.

A *substitution* is a homomorphism from pattern to pattern that maps every constant symbol to itself. For a pattern p and a substitution θ, we denote by $p\theta$ the image of p by θ. The language of a pattern p, written as $L(p)$, is the set $\{w \in \Sigma^+ |$ exists substitution θ such that $w = p\theta\}$. For a set P of patterns, $L(P)$ denotes the language $\{w \in L(p) \mid p \in P\}$ and $\mathcal{L}(P)$ denotes the class of languages $\{L(p) \mid p \in P\}$. A set of patterns P is said to *cover* a set S of strings if $S \subseteq L(P)$.

3 Maximum Density Pattern for a Set of Strings

We consider the problem of finding a set of patterns such that their languages best represent a given set of strings. Since we consider unions of sets of pattern languages, this problem can be compared to those studied in [3, 5]. We introduce a measure for how well a set of patterns cover a given set of strings. For a set of patterns P and a set of strings S, we define the *density of P with respect to S*,

$$d(P, S) = \frac{|L(P) \cap S|}{|L(P) \cap \Sigma^{\leqq \lceil S \rceil}|}.$$

For a set of strings S and two sets of patterns P and Q, both which covers S individually, we argue that P generalizes S better than Q if $d(P, S) > d(Q, S)$, since Q includes more examples not in S within the size of the longest example that is known to exist. For example, in the case where the input strings S is $\{$"abc", "bca"$\}$, "$x_1 bc x_2$" would be chosen over "$x_1 a x_2$", since $d(L(\text{"}x_1 bc x_2\text{"}), S) > d(L(\text{"}x_1 a x_2\text{"}), S)$. Based on this, we introduce the following problem:

Problem Maximum Density Patterns (MDP)

Input: A set S of strings over $\Sigma^{\leqq \lceil S \rceil}$, and a set of sets of patterns $\boldsymbol{\mathcal{P}}$.
Output: A set of patterns $P \in \boldsymbol{\mathcal{P}}$ covering S and maximizing $d(P, S)$.

The choice of the solution space $\boldsymbol{\mathcal{P}}$ fundamentally affects the problem. To ensure that MDP output a target generalization $P \in \boldsymbol{\mathcal{P}}$ on input S, no other generalization $P' \in \boldsymbol{\mathcal{P}}$ covering S should have $d(P', S) \geq d(P, S)$. For example consider the case where $\boldsymbol{\mathcal{P}}$ is the set of all finite sets of patterns from $\boldsymbol{P}$ or $\boldsymbol{RP}$, for which given any input S, an optimal solution is simply S itself. Such an output requires virtually no computation, and probably makes very poor generalization of the input. Ideally, the solution space $\boldsymbol{\mathcal{P}}$ should be such that any $P \in \boldsymbol{\mathcal{P}}$ maximizing $d(P, S)$ is an interesting generalization.

3.1 Computing Density

Consider the computation for an instance of the MDP problem. Since we require that $S \subseteq L(P)$, we have $|L(P) \cap S| = |S|$. To solve MDP with input $\boldsymbol{\mathcal{P}}$ it suffices that we find among all sets of patterns $P \in \boldsymbol{\mathcal{P}}$ for one where: (1) $S \subseteq L(P)$, and (2) $|L(P) \cap \Sigma^{\leqq \lceil S \rceil}|$ is minimized. Condition (1) cannot be computed efficiently if $\boldsymbol{\mathcal{P}} = \{\{p\} \mid p \in \boldsymbol{\mathcal{L}}(\boldsymbol{P})\}$ (refer Section 2 for pattern class $\boldsymbol{P}$), since it is known that the membership problem is NP-complete for the class $\boldsymbol{\mathcal{L}}(\boldsymbol{P})$ [1]. However, for any set of strings S and any $P \subseteq \boldsymbol{RP}$, whether $S \subseteq L(P)$ can be computed in time $O(|P| \cdot \| S \| + |S| \cdot \| P \|)$ [11].

Condition (2) can be solved by counting $|L(P) \cap \Sigma^{\leqq \lceil S \rceil}|$. For regular patterns, counting the number of strings generated by a set of patterns can be done efficiently by constructing a DFA that accepts strings in its languages, and then counting the number of distinct paths of up to a given length it accepts. This gives us the following.

Theorem 1 ([7, 9]). Given a set P of regular patterns and $m \in N^+$, there exists an algorithm which computes the total number of strings in $L(P)$ of length up to m in time $O(m \cdot \| P \| \cdot \min(\{|\Sigma|, \| P \|\}))$. □

3.2 Comparing Densities

Another way to solve Condition (2) is by comparing the number of strings generated. In this section we show how this can be done very efficiently for any two substring patterns of the same length.

For substring patterns, there is a recurrence relation which allows us to count the strings using a dynamic programming approach. The following algorithm (*CountStrings*) computes, given as input a set of strings T and $m \in N^+$, the total number of unique strings of length up to m within $L(P(T))$. We assume that no string in T is a substring of some other string in T. Otherwise, the strings that are substring of some other string in T can be removed in time $O(|T|^2 \lceil T \rceil)$, and their removal will not affect the computation of $|L(P(T)) \cap \Sigma^{\leqq \lceil S \rceil}|$.

CountStrings(T, m)
1. For each r from 1 to m,
2. For each string $t \in T$ where $|t| \leq r$,
3. For each r' from $r - |t| + 1$ to $r - 1$, calculate $S(T, t, r, r')$.
4. Calculate $\Phi(T, r)$.
5. Output $\sum_{r=1}^{m} \Phi(T, r)$.

For each $r \in N^+$, $\Phi(T, r)$ is the total number of strings of length r in $L(P(T))$. For $t \in T$, r, $r' \in N^+$ where $r > r'$, $S(T, t, r, r')$ is the number of strings in $L(\text{“}x_1 t\text{”})$ of length r which does not begin with some string of length r' in $L(P(T))$. It is easy to see that $S(T, t, r+|t|, r) = |\Sigma|^{r-|t|} - \Phi(T, r)$. The following is the computation of $S(T, t, r, r')$. For a string t and $n \in N$, prefix(t, n) denotes the prefix of t of length n, while suffix(t, n) denotes the suffix of t of length $|t|-n$.

$S(T, t, r, r')$
1. If $|t| > r$, output 0.
2. Else if $|t| = r$, output 1.
3. Otherwise, for each $t' \in T$, we try to find all $0 < n < |t'|$ such that
4. (a) suffix(t', n) matches some prefix of t,
5. (b) $n + |t| \leq r$,
6. (c) $|t| - (|t'| - n) \geq r - r'$,
7. and collect all such prefix(t', n) into a set $A_{t,r,r'}$.
8. We then remove from $A_{t,r,r'}$ all the strings for which some of its suffixes
9. are also in $A_{t,r,r'}$. (Hence $A_{t,r,r'}$ is a set of shortest prefix strings of T
10. where the suffix part is consistent with t.)
11. Let $B_{t,r,r'} = \{(t', \text{suffix}(t', n)) \mid \text{prefix}(t', n) \in A_{t,r,r'}\}$.
12. Output $|\Sigma|^{r-|t|} - \Phi(T, r-|t|) - \sum_{(t',t'') \in B_{t,r,r'}} S(T, t', r-|t|+|t''|, r-|t|)$.

For each r, the following is the computation of $\Phi(T, r)$.

$\Phi(T, r)$
1. Initialize ϕ to $|\Sigma| \cdot \Phi(T, r-1)$ if $r \geq 1$, 0 otherwise.
2. For each $t \in T$ where $|t| \geq r$, add $S(T, t, r, r-1)$ to ϕ.
3. Output ϕ.

Theorem 2. Given a set $P(T)$ of substring patterns and $m \in N^+$, $m \gg \lceil T \rceil$, *CountStrings* computes the total number of strings in $L(P(T))$ of length up to m in time $O(m \cdot \| T \|^2)$. □

Note that its runtime is comparable to the algorithm in Theorem 1 for the cases where $|\Sigma|$ is large. A straightforward analysis of the recurrence relation in *CountStrings* gives us the following.

Lemma 1. *Let* $|\Sigma| \geq 3$, $x \in N$ *and* $d \in N^+$. *For all* t, $t' \in \Sigma^*$ *where* $|t| = |t'| > d$, $S(\{t\}, t, x+d, x) > \sum_{0<i<d} S(\{t'\}, t', x+i, x)$.

Lemma 2. *Let* $|\Sigma| \geq 4$, $x \in N$ *and* $d \in N^+$. *For all* t, t', $t'' \in \Sigma^*$ *where* $|t| = |t'| = |t''| > d$, $|S(\{t\}, t, x+d, x) - S(\{t'\}, t', x+d, x)| \leq S(\{t''\}, t'', x+1, x)$.

Since $|T| = 1$, for all t, r and r', for all (t, t_1), $(t, t_2) \in B_{t,r,r'}$, $|t_1| = |t_2|$ implies $t_1 = t_2$. That is, only the length of the overlap is needed. Let $\mathbf{B}(t, r) = \{|t''| \mid (t, t'') \in B_{t,r,r-1}\}$. It is clear that $\max(\mathbf{B}(t, r)) < |t|$. Note that for any string t and $r \geq 2|t| - 1$, $\mathbf{B}(t, r+1) = \mathbf{B}(t, r)$. For this reason we also let $\mathbf{B}(t)$ denote $\mathbf{B}(t, 2|t| - 1)$.

Theorem 3. *Let* $|\Sigma| \geq 4$. *For all* t, $t' \in \Sigma^*$ *where* $|t| = |t'|$, *if* $\mathbf{B}(t) = \mathbf{B}(t')$, *then for all* $r \in N$, $\Phi(\{t\}, r) = \Phi(\{t'\}, r)$. *Otherwise without loss of generality let* $\max(\mathbf{B}(t) \setminus \mathbf{B}(t')) < \max(\mathbf{B}(t') \setminus \mathbf{B}(t))$, *in which case*

(1) $\Phi(\{t\}, r) = \Phi(\{t'\}, r)$ *for* $r < 2|t| - \max(\mathbf{B}(t') \setminus \mathbf{B}(t))$, *and*
(2) $\Phi(\{t\}, r) > \Phi(\{t'\}, r)$ *for* $r \geq 2|t| - \max(\mathbf{B}(t') \setminus \mathbf{B}(t))$.

From Theorem 3, it is clear that for any strings t, t' of the same length, it suffices that we compute the sets $\mathbf{B}(t)$ and $\mathbf{B}(t')$ to decide if $CountStrings(\{t\}, m) > CountStrings(\{t'\}, m)$ for any $m \in N^+$.

4 Time Complexities of the MDP Problem

In this section we study the runtime complexities for MDP for three related cases. We examine the classes derived from unions of languages in $\mathcal{L}(Psub_l)$ for some $l \in N^+$. That is, the languages generated from substring patterns of a fixed length l. We are interested in both the case where the number of languages allowed in a union is bounded, and the case where it is unbounded.

We first look at the case where no union is allowed.

4.1 The Case of $\mathcal{P} = \{\{p\} \mid p \in Psub_l\}$

Theorem 4. *Given* $l \in N^+$ *and* Σ *where* $|\Sigma| \geq 4$. *Let* $\boldsymbol{\mathcal{P}} = \{\{p\} \mid p \in \boldsymbol{Psub}_l\}$ *and* $S \subseteq \Sigma^*$. *There exists an algorithm that solves the* MDP *problem in time* $O(\min(\{\| S \|, |\Sigma|^l\}) \cdot (l^2 + l|S| + \| S \|))$.

Proof. We first consider the total possible candidates of length l. Since ideally we want each pattern in the solution to cover some strings in S, we choose only substrings of strings in S for candidates. There are possibly $\| S \|$ such substrings, of which at most $|\Sigma|^l$ are unique. This gives us a total of $\min(\{\| S \|, |\Sigma|^l\})$ candidates. We denote this candidate set by C. It is easy to verify, using Theorem 3, that the following algorithm ($SolveMDP$) find a set from among C which fulfills the requirement of the MDP problem.

$SolveMDP(C, S)$
1. Let $maxSet = \{-1\}$.
2. For each candidate t in C where $S \subseteq L(P(\{t\}))$
3. Compute $\mathbf{B}(t)$. If $\max(\mathbf{B}(t) \setminus maxSet) > \max(maxSet \setminus \mathbf{B}(t))$ and $\lceil S \rceil \geq 2|t| - \max(\mathbf{B}(t) \setminus maxSet)$, set $maxSet$ to $\mathbf{B}(t)$ and let $p = \{t\}$.
4. If no such t exists, no solution exists. Otherwise output p.

As discussed in Section 3.1, each check at line-2 takes $O(\| S \| + l|S|)$ time and each computation of line-3 takes $O(l^2)$ time, that is, time needed for finding $\mathbf{B}(t)$. Each of these is performed at most once for each candidate in C, thus giving us the runtime as stated. □

It is worth noting that in this case, a solution for MDP can be obtained in time that is not directly dependent on the maximum length of the samples.

Corollary 1. *Given* Σ *where* $|\Sigma| \geq 4$. *Let* $\boldsymbol{\mathcal{P}} = \{\{p\} \mid p \in \boldsymbol{Psub}\}$ *and* $S \subseteq \Sigma^*$. *There exists an algorithm that solves the* MDP *problem in time* $O(\| S \| \cdot (\lfloor S \rfloor^3 + \lfloor S \rfloor^2 |S| + \lfloor S \rfloor \| S \|))$.

Proof. To find a solution in this case it suffices that we compute MDP for $\boldsymbol{\mathcal{P}} = \{\{p\} \mid p \in \boldsymbol{Psub}_l\}$ for each $l \leq \lfloor S \rfloor$. □

We next look at the case where the unions can involve at most k languages, for some $k \in N^+$, $k \geq 2$.

4.2 The Case of $\boldsymbol{\mathcal{P}} = \{\boldsymbol{P} \subseteq \boldsymbol{Psub_l} \mid |\boldsymbol{P}| \leq \boldsymbol{k}\}$

Theorem 5. Given l, $k \in N^+$, $k \geq 2$ and Σ where $|\Sigma| \geq 2$. Let $\boldsymbol{\mathcal{P}} = \{P \subseteq \boldsymbol{Psub}_l \mid |P| \leq k\}$ and $S \subseteq \Sigma^*$. There exists an algorithm that solves the MDP problem in time $O(\min(\{\| S \|, |\Sigma|^l\})^k \cdot (\lceil S \rceil \cdot lk \cdot \min(\{|\Sigma|, lk\}) + lk|S| + k \| S \|)$.

Proof. The proof is the same as that for Theorem 4 except that the number of candidates in this case is $\min(\{\| S \|, |\Sigma|^l\})^k$, and the check at line-3 is not valid. Hence in this case, we use the string counting algorithm as in Theorem 1 for the comparison at line-3. Each computation is done in time $\lceil S \rceil \cdot lk \cdot \min(\{|\Sigma|, lk\})$, giving us the runtime as stated. □

We next consider the computational complexity of MDP with respect to unbounded unions of the languages in $\mathcal{L}(\boldsymbol{Psub}_l)$. Regrettably, in this case, the MDP problem turns out to be NP-complete.

4.3 The Case of $\boldsymbol{\mathcal{P}} = \boldsymbol{Psub}_l^*$

We first consider the case where l is very small. In this case, the problem can be solved in time polynomial in the size of the input as follows. Let $T = \{s \mid |s| = l$ and s is a substring of some $w \in S\}$. Hence $|T| \leq |\Sigma|^l$. We then check if $S \subseteq L(P)$ for each $P \subseteq P(T)$, compute its density, and find such P with the highest density. Each computation for T is done in $O(\lceil S \rceil l|T| \min(\{|\Sigma|, l|T|\}))$ by Theorem 1, and the number of candidates T is bounded by $2^{|\Sigma|^l}$. In total, all the computation can be done in $O(2^{|\Sigma|^l} \lceil S \rceil l^2 |\Sigma|^{l+1})$ time.

Theorem 6. The problem of MDP for $\boldsymbol{Psub}_l^*$ can be solved in polynomial time, if l is $O(poly(\log\log(\max\{|S|, \lceil S \rceil\}))$. □

Unfortunately, this problem is NP-hard in general. More precisely, we can show the NP-completeness of the decision version of MDP for $\boldsymbol{Psub}_l^*$; for input parameter D, does there exist a set P of patterns in $\boldsymbol{\mathcal{P}}$ such that $S \subseteq L(P)$ and $d(P, S) \geq D$?

Theorem 7. MDP for $\boldsymbol{Psub}_l^*$ (decision version) is NP-complete.

To prove this, we infer on the number of strings of length m in $|L(P)|$ where $P \in \boldsymbol{Psub}_l^*$. We obtain the following bounds on this number for any P consisting solely of non-overlapping strings from the inclusion-exclusion principle.

Lemma 3. Let T and T' be sets of non-overlapping strings of length l. If $m \leq l^2$, $|T'| = |T| + 1 \leq l$, $|\Sigma| \geq 3$ and $l \geq 14$, then $|L(P(T')) \cap \Sigma^{=m}| > (m - l + 1)|\Sigma|^{m-l} > |L(P(T)) \cap \Sigma^{=m}|$ holds. □

The Proof of Theorem 7. Obviously, the decision version of MDP for $\boldsymbol{Psub}_l^*$ belongs to NP by Theorem 1.

We then show the completeness part. We reduce a well-known NP-complete problem, MINIMUM 3-SET COVER (for short, M3SC) [8] to our MDP.

Problem MINIMUM 3-SET COVER	
Input:	A set $U = \{1, \ldots, n\}$, its subsets $V_1, \ldots, V_k \subseteq U$, where $\lvert V_i \rvert = 3$ for $i = 1, \ldots, k$, and an integer $K > 0$.
Question:	Is there a family $\mathcal{V} \subseteq \{V_1, \ldots, V_m\}$ of subsets, such that $\bigcup_{V_i \in \mathcal{V}} V_i = U$ and $\lvert \mathcal{V} \rvert \leq K$?

For the M3SC instance, we construct a set of strings $S \subseteq \Sigma^{=m}$, where $\Sigma = \{a, b, c\}$, $m = 3k(r+2)$ with $r = \max\{n, k\}$, and let $l = 2(r+2)$. In the construction, we compose $3k$ component (sub)strings with length $l = r+2$ into strings with length $m = 3k(r+2)$. For this purpose, we first define component strings. For each subset V_i, we define string $v^{(i)} \in \Sigma^{=r+2}$ as follows:

$$v^{(i)} = \overbrace{c\underbrace{bb\cdots b}_{i}a\underbrace{bb\cdots b}_{i}ab\cdots ba\underbrace{bb\cdots b}_{p}}^{r+2},$$

where $p \equiv r+1 \pmod{i+1}$. For example, $v^{(1)} = cbaba\cdots$, $v^{(2)} = cbbabba\cdots$. Also, for each element $i \in U$ we define three strings $u^{(i,\alpha,\beta)}$:

$$u^{(i,\alpha,\beta)} = \overbrace{\alpha\underbrace{a\cdots a}_{p}b\cdots ab\underbrace{aa\cdots a}_{i}b\underbrace{aa\cdots a}_{i}b\beta}^{r+2},$$

where α, $\beta \in \{a, b, c\}$ and $p \equiv r \pmod{i+1}$. For example, $u^{(3,a,c)} = aa\cdots abaaabaaac$. By using these component strings in $R = \{v^{(i)}, \text{ for } i = 1, \ldots, k\} \cup \{u^{(i,\alpha,\beta)}, \text{ for } i = 1, \ldots, n, \text{ and } \alpha, \beta \in \{a, b, c\}\}$, we now construct a set S of strings corresponding to the base set U of a given M3SC problem instance. Each string $w \in S$ consists of $3k$ substrings of length $(r+2)$, each of which is a string in R. Let us denote the j-th substring of string w by w_j. For an element $i \in U$, the corresponding string $w^{(i)}$ is defined as follows:

$$w_h^{(i)} = \begin{cases} u^{(i,\alpha,\beta)} & \text{if } h \equiv 0 \pmod 3 \\ v^{(j)} & \text{if } h \not\equiv 0 \pmod 3 \text{ and } i \in V_j, \text{ where } j = \lceil h/3 \rceil \\ u^{(i,c,c)} & \text{otherwise.} \end{cases}$$

where

$$\alpha = \begin{cases} a & \text{if } i \in V_j \text{ and } \lvert\{1, 2, \ldots, i-1\} \cap V_j\rvert = 0, \\ b & \text{if } i \in V_j \text{ and } \lvert\{1, 2, \ldots, i-1\} \cap V_j\rvert = 1, \\ c & \text{otherwise,} \end{cases}$$

$$\beta = \begin{cases} a & \text{if } i \in V_{j+1} \text{ and } \lvert\{1, 2, \ldots, i-1\} \cap V_{j+1}\rvert = 0, \\ b & \text{if } i \in V_{j+1} \text{ and } \lvert\{1, 2, \ldots, i-1\} \cap V_{j+1}\rvert = 1, \\ c & \text{otherwise.} \end{cases}$$

For example, if in the original M3SC instance the sets to which element 4 belongs are only $V_1 = \{4, 5, 6\}$ and $V_3 = \{1, 4, 6\}$, the corresponding string is

$$w^{(4)} = \underbrace{v^{(1)}v^{(1)}u^{(4,a,c)}}_{4 \in V_1}\underbrace{u^{(4,c,c)}u^{(4,c,c)}u^{(4,c,b)}}_{4 \notin V_2}\underbrace{v^{(3)}v^{(3)}u^{(4,b,c)}}_{4 \in V_3}u^{(4,c,c)}\cdots u^{(4,c,c)}\cdots.$$

Note that each segment $u^{(i,\alpha,\beta)}$ is unique to $w^{(i)}$ and only $v^{(j)}$s may appear in more than one string in S. Now we claim that for this string set $S = \{w^{(i)}, i \in U\}$, the maximum density of $P \in \boldsymbol{Psub}_l^*$ is not smaller than $n/(K \cdot \sum_{j=l}^{m}((j-l+1)3^{j-l}))$, if and only if the M3SC's answer is yes.

We prove the only-if part. Let $P(T') \in \boldsymbol{Psub}_l^*$ be a cover for S with the highest density. By the above argument, every $t \in T'$ must be a substring of some $s \in S$. Here we have the following three types of substrings t with length $l = 2(r+2)$:

1) $t = v^{(i)}v^{(i)}$, where $i \in \{1, 2, \ldots, m\}$,
2) t contains $u^{(j,\alpha,\beta)}$ as its substring where $j \in \{1, \ldots, n\}$,
3) $t = \overbrace{a\cdots abaa\cdots\beta}^{q} v^{(i)} \overbrace{cbb\cdots babb\cdots}^{r+2-q}$, or $\overbrace{b\cdots babb\cdots}^{q} v^{(i)} \overbrace{\alpha aa\cdots abaa\cdots}^{r+2-q}$, where $\alpha, \beta \in \{a, b, c\}$ and $i \in \{1, 2, \ldots, m\}$.

Type 1) and 2) of substring pattern covers three strings and one string in S, respectively. As for Type 3), t covers just one string in S, since all the three strings which contain $v^{(i)}$ have different α or β. From these, it follows that when we maximize the density of patterns to cover S, only Type 1) strings are in T'. For solution P with only Type 1) strings, its density is $d(P,S) = \frac{|L(P)\cap S|}{|L(P)\cap \Sigma^{\leq m}|} = \frac{|S|}{\sum_{i=k}^{m}|L(P)\cap\Sigma^{=i}|}$, where $m = 3k(r+2)$ and $l = 2(r+2)$. By the construction of $v^{(i)}$, P with only Type 1) strings are non-overlapping, $m = 3k(r+2) \leq l^2$ and $|P| \leq k \leq l$. By these properties and Lemma 3, solutions P with $|P| = K$ and P' with $|P'| = K+1$ satisfy that

$$d(P,S) = \frac{|S|}{\sum_{i=l}^{m}|L(P)\cap\Sigma^{=i}|} \geq \frac{n}{\sum_{j=l}^{m}(K(j-l+1)\cdot 3^{j-l})} \tag{1}$$

$$\geq \frac{|S|}{\sum_{i=l}^{m}|L(P')\cap\Sigma^{=i}|} = d(P',S). \tag{2}$$

Inequality (1) implies that, if the optimal solution is not smaller than $n/(K \cdot (\sum_{j=l}^{m}((j-l+1)\cdot 3^{j-l}))$, $K \geq |P|$ holds. Recall that each member of P is Type 1) and forms $x_1v^{(i)}v^{(i)}x_2$, which corresponds to set V_i. Thus, by solving MDP for $\boldsymbol{Psub}_l^*$, we can obtain the solution of M3SC, which proves the only-if part. The opposite direction can also be shown by a similar argument and (2). □

5 Conclusions and Future Work

In this paper, we considered the problem of finding patterns that covers a given set of strings. We have argued in this paper our dissatisfactions with current strategies to avoid over-generalizations in the solution pattern languages. We proposed a natural measure, "density", to evaluate the degree of over-generalization of a given solution, and considered the problem of finding among all solutions for one which maximizes such density.

We showed that the resultant problem is closely related to the counting problem and the membership problem. As such, we listed methods to perform these efficiently for a few cases where such computations are possible. We showed that for substring patterns of the same length, to compare between the number of strings up to a certain length generated by any two patterns does not require an exhaustive counting of the strings. It remains to be seen if similar methods exist for other subclasses of the pattern languages.

Finally, we studied the runtime complexities for solving the MDP problem on three related classes of languages derived from unions of the substring pattern languages. One of these turned out to be very efficient, the other less so, but still within runtime polynomial to the input size. Regrettably, for the third class of languages we considered, the set of unbounded unions of fixed-length substring pattern languages, the problem of finding a solution which maximizes the density can be related to the Minimum Set Cover Problem, and is shown to be NP-complete.

Acknowledgement

The authors would like to thank Prof. Takeshi Shinohara for discussions.

References

1. D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.
2. H. Arimura, R. Fujino, T. Shinohara, and S. Arikawa. Protein motif discovery from positive examples by Minimal Multiple Generalization over regular patterns. In *Proceedings of the Genome Informatics Workshop*, pages 39–48, 1994.
3. H. Arimura, T. Shinohara, and S. Otsuki. Finding minimal generalizations for unions of pattern languages and its application to inductive inference from positive data. In *Proc. Annual Symp. on Theoretical Aspects of Computer Sci.*, 1994.
4. A. Brāzma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *J. Comp. Biol.*, 5(2):277–304, 1998.
5. A. Brāzma, E. Ukkonen, and J. Vilo. Discovering unbounded unions of regular pattern languages from positive examples. In *Proceedings of the 7-th International Symposium on Algorithms and Computation (ISAAC'96)*, 1996.
6. B. Brejova, T. Vinar, and M. Li. *Pattern Discovery: Methods and Software*, chapter 29, pages 491–522. Humana Press, 2003.
7. Chee-Yong Chan, Minos Garofalakis, and Rajeev Rastogi. RE-tree: an efficient index structure for regular expressions. *The VLDB Journal*, 12(2):102–119, 2003.
8. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., New York, 1979.
9. Sampath Kannan, Z. Sweedyk, and Steve Mahaney. Counting and random generation of strings in regular languages. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 551–557. Society for Industrial and Applied Mathematics, 1995.

10. M. Sato, Y. Mukouchi, and D. Zheng. Characteristic sets for unions of regular pattern languages and compactness. In *Algorithmic Learning Theory: Ninth International Conference (ALT' 98)*, volume 1501 of *Lecture Notes in Computer Science*, pages 220–233. Springer-Verlag, 1998.
11. T. Shinohara. Polynomial time inference of extended regular pattern languages. In *RIMS Symposia on Software Science and Engineering, Kyoto, Japan*, volume 147 of *Lecture Notes in Computer Science*, pages 115–127. Springer-Verlag, 1982.
12. J. Uemura and M. Sato. Compactness and learning of classes of unions of erasing regular pattern languages. In *Algorithmic Learning Theory: Thirteenth International Conference (ALT' 02)*, volume 2533, pages 293–307. Springer-Verlag, 2002.

String Coding of Trees with Locality and Heritability

Saverio Caminiti and Rossella Petreschi

Dipartimento di Informatica, Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
{caminiti,petreschi}@di.uniroma1.it

Abstract. We consider the problem of coding labelled trees by means of strings of vertex labels and we present a general scheme to define bijective codes based on the transformation of a tree into a functional digraph. Looking at the fields in which codes for labelled trees are utilized, we see that the properties of locality and heritability are required and that codes like the well known Prüfer code do not satisfy these properties. We present a general scheme for generating codes based on the construction of functional digraphs. We prove that using this scheme, locality and heritability are satisfied as a direct function of the similarity between the topology of the functional digraph and that of the original tree. Moreover, we also show that the efficiency of our method depends on the transformation of the tree into a functional digraph. Finally we show how it is possible to fit three known codes into our scheme, obtaining maximum efficiency and high locality and heritability.

1 Introduction

Labeled trees are of interest in both practical and theoretical areas of computer science. To take just two examples: Ethernet has a unique path between terminal devices, labeling the tree vertices is necessary to identify each device in the network without ambiguity; trees are used in biology to represent phylogenetic relationships between species, populations, individuals, or genes represented by labels.

Coding labeled trees by means of strings of vertex labels is an interesting alternative to the usual representations of tree data structures in computer memories, since it has many practical applications [3]. Evolutionary algorithms over trees maintain a population of data structures that represents candidate solutions to a problem. The association between structures and solutions is realized through a decoder which must exhibit efficiency, locality, and heritability if the evolutionary search is to be effective [4, 5, 7, 12]. In this context it is possible to show that representing a tree as a string increases the probability to guarantee the required properties will be attained.

Furthermore, string base coding makes it possible to generate random uniformly distributed trees and random connected graphs [9]. Indeed the generation of a random string, followed by the use of a fast decoding algorithm, is typically

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 251–262, 2005.

more efficient than generating a tree by adding edges randomly, where one must be careful not to introduce cycles.

Finally, tree codes are also used for data compression and in the computation of forest volumes of graphs [8].

Unless stated otherwise, here we will consider the tree as rooted in vertex 0 and its n vertices labeled from 0 to $n-1$.

The *naïve* method for relating a tree to a string P consists in associating to each vertex x the value of its parent $p(x)$; P has cardinality $n-1$ since the root node 0 can be omitted, in the following we refer to the naïve string as parent array. It should be noted that an arbitrary string of length $n-1$ over $[0, n-1]$ is not necessarily a tree, but it may be either a non-connected or a cyclic graph.

We are interested in those types of coding that define a bijection between the set of labeled trees of n vertices and a set of strings over $[0, n-1]$. Since Cayley has proved that the number of labeled trees on n vertices is n^{n-2}, we know that this kind of one-to-one correspondence requires the cardinality of the string to be equal to $n-2$ [1].

In his proof of Cayley's theorem, Prüfer provided the first bijective string based coding for trees [11]. Over the years since then many codings behaving like that of Prüfer have been introduced. In [2] a complete survey on these codes is presented, and it is shown that coding and decoding in sequential linear time is possible using each of these codes; efficient parallel algorithms are also presented.

However, even if they are extremely efficient, Prüfer-like codes lack other desirable properties, such as locality and heritability, as noted in [6]. An experimental analysis [7] shows that these properties are much better satisfied by the Blob code, defined by Picciotto [10].

In this paper we present a general scheme for defining bijective codes based on the transformation of a tree into a functional digraph. We also show how the properties of locality and heritability are related to differences between the digraph and the original tree. Then we highlight the differences between Prüfer-like codes and codes derivable from our scheme. Finally, we show how it is possible to map some known codes, including the Blob code, to our scheme.

2 Preliminaries

In this section, we introduce some definitions that will be needed in the rest of the paper.

Definition 1. *Given a function g from the set $[0, n]$ to the set $[0, n]$, the* functional digraph $G = (V, E)$ *associated with g is a digraph with $V = \{0, \ldots, n\}$ and $E = \{(v, g(v))$ for every $v \in V\}$.*

For this class of graphs the following lemma holds:

Lemma 1. *A digraph $G = (V, E)$ is functional if and only if $|E| = n$ and the outer degree of each vertex is equal to 1.*

Corollary 1. *Each connected component of a functional digraph is composed of several trees, each of which is rooted in a vertex belonging to the* core *of the component, which is either a cycle or a loop (see Figure 1a).*

Functional digraphs are easily generalizable for the representation of functions which are undefined in some values: if $g(x)$ is not defined, the vertex x in G does not have any outgoing edge. The connected component of G containing an x, such that $g(x)$ is not defined, is a tree rooted in x without cycles or loops (see Figure 1b).

Definition 2. *A* labeled n-tree *is an unrooted tree with n vertices, each with a distinct label selected in the set* $[0, n-1]$.

Definition 3. *In a labeled n-tree, the set of vertices between a vertex v to a vertex u is called the* path *from v to u; u and v do not belong to the path.*

Since in this paper we deal only with labeled trees, we will refer to them simply as trees. In the following, when it is necessary to root a tree in one of its vertices, we will consider its edges oriented upwards from leaves to root.

Remark 1. Let T be a rooted tree and $p(v)$ be the parent of v for each v in T. T is the functional digraph associated with the function p.

Let us call *n-string* a string of n elements in the set $[0, n+1]$.

Definition 4. *A code is a method for associating trees to strings in such a way that different trees yield different strings. A* bijective code *is a code associating n-trees to $(n-2)$-strings.*

Below, when there is no risk of confusion, we will identify a tree with its associated string, and vice versa.

A code satisfies the *Locality Property* if small changes in the tree correspond to small changes in the associated string, and vice versa.

In evolutionary algorithms, where sometimes a new string is generated by mixing two existing strings, another desirable property is the *Heritability Property*: edges of the tree corresponding to the mixed string belong to one of the two existing trees.

Let us look at the naïve code representing a tree with the parent vector. Since each edge of a tree corresponds to an element of the string, this code exhibits maximal locality: a single change in the tree corresponds to a single change in the associated string, and vice versa (see Figure 1c and 1d). Naïve code also maximally satisfies heritability: in each string the i-th element corresponds to the edge $(i, p(i))$ of the tree, it implies that a tree obtained by mixing two existing strings has only edges coming from the two existing trees. Unfortunately, this code is not bijective, so a string obtained by modifying one or more strings is not necessarily a tree: more precisely, the probability of obtaining a tree is $\frac{1}{n}$. This is a serious shortcoming of naïve code.

The *Prüfer* code proceeds recursively, deleting the leaf with smallest label from the tree; when a leaf is deleted, the label of its parent is added to the

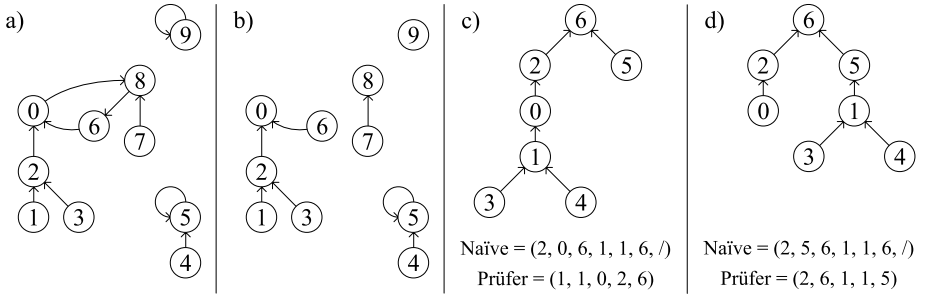


Fig. 1. a) A functional digraph associated with a fully defined function; b) A functional digraph associated with a function undefined in 0, 8, and 9; c) A tree T, and the corresponding naïve and Prüfer codes (notice that this tree is not rooted at 0 and then the naïve code has cardinality n); d) $T' = T - (1, 0) + (1, 5)$ and the corresponding naïve and Prüfer codes.

code. This code is bijective, but exhibits extremely poor locality [6] (see Figure 1c and 1d).

3 General Method

In this section we present a general method for defining bijections between the set of labeled n-trees and $(n-2)$-strings. Our idea is to modify the naïve method so as to reduce the dimension of the string that it yields.

In order to build an $(n-2)$-string, we conjecture that the tree is rooted at a fixed vertex x, and that there exists another fixed vertex y having x as parent. Under these assumptions, in the parent array representation we may omit the information related to both x and y. It is easy to root a given unrooted tree at a fixed vertex x, however it is not so clear how to guarantee the existence of edge (x, y). A function φ manipulates the tree in order to ensure the existence of (x, y) and this is what characterizes each specific instance of our general method. The function φ has to transform T into a functional digraph G, with $n-1$ edges associated with a function g, such that $g(x)$ is undefined and $g(y) = x$. Below we see the coding scheme when φ, x, and y are known:

General Coding Scheme
Input: Input: an n-tree T
Output: Output: an $(n-2)$-string C
1. Root T in x
2. Construct $G = \varphi(T)$
3. **for** $v = 0$ **to** $n-1$ **do**
4. **if** $(v \neq x$ **and** $v \neq y)$ **then** add $g(v)$ to C

To guarantee the bijectivity of the coding obtained, the function φ must be invertible; only under this hypothesis is it possible to define the decoding scheme:

General Decoding Scheme
Input: Input: an $(n-2)$-string C
Output: Output: an n-tree T

1. Reconstruct the graph G starting from code C
2. Add the fixed edge (x, y)
3. Compute $T = \varphi^{-1}(G)$

In our method, the topology of graph G directly identifies the string C, since for each vertex from 0 to $n-1$ its outgoing edge is considered. The obtained C is similar to the naïve code of the tree to precisely the same extent as the tree topology is similar to the graph topology. Since the naïve code has naturally maximal locality and heritability, if we are interested in obtaining high locality and heritability codes we have to look for those φ functions that minimize the variations introduced into the tree. Consequently the efficiency of our coding and decoding schemes is strictly dependent on the computation of φ.

It should be noted that in all Prüfer-like codes the tree topology determines the elimination order of vertices, so a small change in the tree may cause a variation of this order and thus a big change in the string (see Figure 1c and 1d). This is the reason why Prüfer and Prüfer-like codes exhibit low locality and heritability [6].

In the following, we show that several codes introduced in the literature [10] can be mapped into our general scheme, and we provide optimal computation for their φ functions.

4 Blob Code

The *Blob* code was introduced by Picciotto [10] in her Ph.D. Thesis. The algorithm used to obtain a string starting from a tree is:

Blob Coding Algorithm
Input: Input: an n-tree T
Output: Output: an $(n-2)$-string C

1. Initialize $blob = \{n\}$, $C = ()$
2. Root T in 0
3. **for** $v = n-1$ **to** 1 **do**
4. **if** $((path(v, 0) \bigcap blob) \neq \emptyset)$ **then** $C[v-1] = p(v)$
5. delete $(v, p(v))$ and insert v in $blob$
6. **else** $C[v-1] = p(blob)$
7. delete $(blob, p(blob))$ and add $(blob, p(v))$
8. delete $(v, p(v))$ and insert v in $blob$

In this algorithm *blob* is a macro-vertex, i.e. it has a parent but it contains many other vertices. Each vertex included in *blob* maintains its own subtree, if any, but this subtree is not necessarily included in the *blob*.

We will call *stable* all vertices satisfying the test in line 4; their corresponding value in the code is their original parent.

Analyzing this algorithm we can see that the line 4 condition is not tied to the incremental construction of the *blob*, but it can be globally computed in the initialization phase as the Lemma 2 asserts:

Lemma 2. *Stable vertices are all vertices v such that $v < max(path(v, 0))$.*

Proof. At step v of main cycle the set *blob* contains all the vertices from $v + 1$ to n. Then the condition of line 4 holds if and only if at least a vertex greater than v occurs in $path(v, 0)$.

Note that $path(v, 0)$ is a set as stated in Definition 3.

Lemma 3. *For each unstable vertex v, $p(z)$ is the corresponding value in the code, where z is $min\{u|u > v$ and u unstable$\}$.*

Proof. In line 6 the current parent of *blob* defines the code value corresponding to an unstable vertex v and in line 7 the *blob* becomes child of $p(v)$. It implies that when line 6 is executed for vertex v, $p(blob)$ is equal to the parent of the smaller unstable vertex greater than v, i.e. $p(z)$.

Let us define a function φ_b constructing a graph G starting from a tree T in the following way: for each unstable vertex v, removes edge $(v, p(v))$ and add edge $(v, p(z))$ where $z = min\{u|u > v$ and u unstable$\}$. If z does not exist, i.e. when $v = n$, add the edge $(v, 0)$.

In Figure 2a and 2b a tree T and a graph $G = \varphi_b(T)$ are depicted.

Remark 2. Each path in T from a stable vertex v to $m = max(path(v, 0))$ is preserved in $G = \varphi_b(T)$.

Theorem 1. *It is possible to fit Blob code into our general scheme when $x = 0$, $y = n$, and $\varphi = \varphi_b$.*

Proof. It is trivial to see that graph $G = \varphi_b(T)$ is a functional digraph, since: a) each vertex has outdegree equal to 1; b) the function g associated with G is undefined in 0; c) $g(n) = 0$.

Lemmas 2 and 3 guarantee that the generated string C is equal to the code computed by Blob Coding Algorithm.

Now we have to prove that φ_b is invertible, i.e. we have to show how to rebuild T from G.

First we eliminate cycles from G, then we recompute stable and unstable vertices of original T to identify, according to Remark 2, those vertices that must recompute their parents in G.

Each cycle Γ is broken deleting the edge outgoing from γ, the maximum label vertex in Γ. Remark 2 implies that γ was unstable in T, indeed if γ was stable in T the path from γ to $max(path(\gamma, 0))$ must appear in G, but this implies a vertex greater than γ in Γ. Notice that γ becomes the root of its own connected

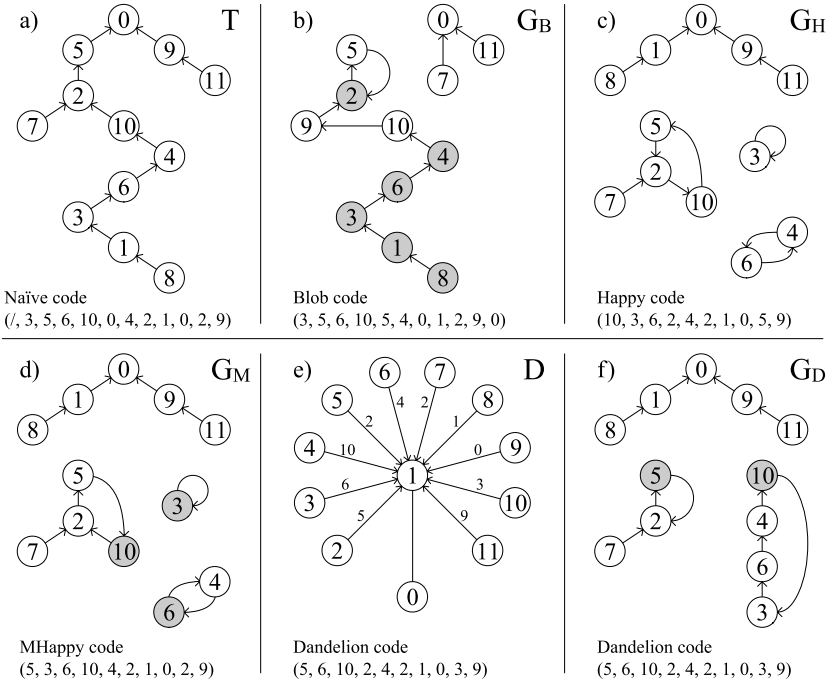


Fig. 2. a) A sample tree T rooted in 0; b) $G_B = \varphi_b(T)$, stable vertices are represented in gray; c) G_H computed from T by the original Happy Coding Algorithm; d) $G_M = \varphi_m(T)$, maximal vertices are represented in gray; e) D computed from T by the Dandelion Coding Algorithm; f) $G_D = \varphi_d(T)$, flying vertices are represented in gray.

component, while 0 is the root of the only connected component not containing cycles. The identification of γ is a step towards the recomputation of stable and unstable vertices.

We call stable in G each vertex v such that $max(path(v, \gamma_v) \bigcup \{\gamma_v\}) > v$, where γ_v is the root of the connected component containing v.

The path preservation stated in Remark 2 guarantees that each vertex v, stable in T, is stable in G. Let us now prove that the vice versa is also true. Let us assume, by contradiction, that there exists a vertex v stable in G but unstable in T. And let us call $m = max(path(v, \gamma_v) \bigcup \{\gamma_v\})$ in G. It holds $v < m$ and m unstable both in G and in T. In G m is unstable because there are not vertices greater than m in $path(v, \gamma_v) \bigcup \{\gamma_v\}$; in T m can not be stable because, as noted before, each stable vertex in T remains stable in G.

W.l.o.g. we assume that all vertices between v and m are stable both in G and in T. Let w be the parent of v in G. By definition of φ_b there exists a vertex $u > v$ unstable in T such that $p(u) = w$ in T. In Figure 3 v, m, u, and w are depicted both in G and in T.

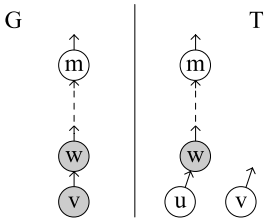


Fig. 3. Vertices involved in the proof of Theorem 1 both in G and in T. Stable vertices are represented in gray.

Since m is in the path from u to 0 in T, m must be smaller than u. Then $v < m < u$ and m is unstable in T contradicting the assertion that there are no unstable vertices in T between v and u (by definition of φ_b).

The computational complexity of original Blob coding and decoding algorithms are quadratic, due to the computation of paths at each iteration. Our characterization of stable vertices (cfr. Lemma 2) decreases the complexity of coding algorithm to $O(n)$. Linear complexity for both coding and decoding can be obtained by fitting Blob code into our general scheme. Indeed both φ_b and φ_b^{-1} can be implemented in $O(n)$ sequential time: computation of the maximum vertex in the upper path (coding) and cycles identification (decoding) can both be implemented by simple search techniques.

In [7], an experimental analysis shows that locality and heritability are satisfied by the Blob code much better than by the Prüfer code. The reasons behind the experimental results become clear when Blob code is analyzed according to our method, which is quite different from Picciotto's original idea. The functional digraph generated by φ_b preserves an edge of the original tree for each stable vertex, and for these vertices $g(v) = p(v)$: this partial similarity with naïve code is the reason for the soundness of locality and heritability.

In the next two sections we discuss two codes that better exploit similarities with naïve code.

5 Happy Code

Happy code was introduced in [10], and it appears with a structure completely different from the Blob code:

Happy Coding Algorithm
Input: Input: an n-tree T
Output: Output: an $(n-2)$-string C
1. Root T in 0 and initialize $J = p(1)$
2. **while** $p(1) \neq 0$ **do**
3. $j = p(1)$, delete $(1, j)$, delete $(j, p(j))$, and add $(1, p(j))$
4. **if** $j > J$ **then** $J = j$ and add (J, J)
5. **else** add $(j, p(J))$, delete $(J, p(J))$, and add (J, j)
6. **for** $v = 2$ **to** n **do** $C[v-2] = p(v)$

This algorithm focuses on the path from 1 to 0. Since the aim of the algorithm is to ensure the existence of edge $(1, 0)$, all the vertices on the original path from 1 to 0 are sequentially moved in order to form cycles. Let us call *maximal* each vertex v in $path(1, 0)$ such that $v > max(path(1, v))$. The first cycle is initialized with $p(1)$ and each time a maximal vertex is analyzed a new cycle is initialized (see Figure 2c).

Notice that the algorithm inserts a vertex j in a cycle immediately after J, the maximal vertex in the cycle. This implies that in the resulting graph the vertices in a cycle will be in reverse order with respect to their position in the original tree (see Figure 2c). Since we are interested in keeping the graph as close as possible to the original tree, we will consider a slightly modified version of this code which avoids this inversion: j is attached immediately before J instead of immediately after. Let us call this modified version of happy code *MHappy* code (see Figure 2d).

For MHappy code we define a function φ_m which, given a tree T, constructs a graph G in the following way: for each maximal vertex v in $path(1, 0)$ remove the edge incoming at v in this path, and add an edge (z, v) where z is the child of the next maximal vertex. If z does not exist, use the child of 0; finally remove the edge incoming at 0 in the path and add the edge $(1, 0)$.

Theorem 2. *It is possible to fit MHappy code into our general scheme when* $x = 0$, $y = 1$, *and* $\varphi = \varphi_m$.

Proof. It is trivial to see that the MHappy coding transforms T into the same functional digraph generated by φ_m: this corresponds to a function g undefined in 0 (the root) and is such that $g(1) = 0$ (edge $(1, 0)$).

To show that φ_m is invertible, first sort all cycles in G into increasing order with respect to their maximum vertex γ, then breack each cycle removing the edge incoming at γ. Since the order of cycles obtained is the same as that in which they where originally created, we rebuild the original tree inserting all the vertices of each cycle in the path from 1 to 0 in accordance with the order of the cycles.

φ_m and φ_m^{-1} can be implemented in $O(n)$ sequential time because coding requires the computation of maximal vertices in the path from 1 to 0 and decoding requires cycle identification and integer sorting. Therefore coding and decoding require linear time both for Happy and MHappy algorithms. φ_m modifies only edges on the path between 1 and 0, so it preserves the topology of T better than φ_b: this improves the level of locality and heritability of this code.

6 Dandelion Code

In the following we present the *Dandelion* code as introduced in [10] with labels on edges:

Dandelion Coding Algorithm

Input: Input: an n-tree T
Output: Output: an $(n-2)$-string C
1. Root T in 0
2. **for** $v = n$ **to** 2 **do**
3. $h = p(v)$, $k = p(1)$, delete (v, h), and add $(v, 1)$ with label h
4. **if** a cycle has been created **then** delete $(1, k)$, add $(1, h)$, $label(v, 1) = k$
5. **for** $v = 2$ **to** n **do** $C[v-2] = label(v, 1)$

The name of dandelion for this code derives from the fact that connecting all the vertices to vertex 1, a tree which looks like a dandelion flower is created (see Figure 2e). Analyzing the algorithm, we can see that the only vertices having the outgoing edge labeled with a value different from their original parent are those verifying the test of line 4, let us call them *flying* vertices.
In code C, a position corresponding to a non-flying vertex v merely displays $p(v)$, showing that the algorithm does not add new information if it considers all the vertices. Hence let us restrict our attention to flying vertices.

Lemma 4. *Flying vertices are all vertices v such that $v \in path(1, 0)$ and $v > max(path(v, 0))$.*

Proof. The first condition trivially holds, otherwise cycles cannot be created.

Given $v \in path(1, 0)$, let $m = max(path(v, 0))$. If $m > v$ then m is processed before v by the algorithm, m is directly connected to 1 and it introduces a cycle containing v. When the cycle is broken (line 4), all the vertices in the cycle are excluded from $path(1, 0)$. This implies that in successive steps v can not be a flying vertex.

On the other hand, if $v > m$ it will be in $path(1, 0)$ when it is processed by the algorithm and so it obviously introduces a cycle.

When a cycle is broken (line 4) in a flying vertex v, 1 will be connected to h (the old parent of v) and the label of edge $(v, 1)$ becomes k (the old parent of 1). In code C, the position corresponding to v displays the value k. Thus, assigning $p(v) = k$, it is possible to avoid edge labels and to generate C directly from p.

Let us define a function φ_d which, given a tree T, constructs a graph G that considers flying vertices of T in decreasing order. For each flying vertex v, φ_d exchange $p(v)$ and $p(1)$ (see Figure 2f).

Theorem 3. *It is possible to fit Dandelion code into our general scheme when $x = 0$, $y = 1$, and $\varphi = \varphi_d$.*

Proof. $G = \varphi_d(T)$ is a functional digraph corresponding to a function g undefined in 0 (the root) and such that $g(1) = 0$ (edge $(1, 0)$). It is also easy to see that the code generated using φ_d is the same as that using Dandelion Coding Algorithm. Considerations similar to those presented in Theorem 2 for φ_m can be used to prove that φ_d is invertible; note that cycles of G must be considered in increasing order of their maximum vertex.

The complexity of the original Dandelion algorithms for coding and decoding is non linear, while it becomes linear when fitted into our general scheme, in view of the fact that φ_d requires the same operations as φ_m.

Concerning locality and heritability, Dandelion code has a behavior which is identical to MHappy code, in spite of the fact that in [10] it was introduced as "a mélange of the methods for Happy code and Blob code". Indeed, both φ_d and φ_m work only on edges in the path between 1 and 0. On this path vertices that modify their parent are maximal (i.e. all vertices v such that $v > max(path(1, v))$) for MHappy code and flying (i.e. all vertices v such that $v > max(path(v, 0))$) for Dandelion code.

7 Conclusion and Open Problems

In the introduction, we stated that we where looking for codes satisfying properties like efficiency, locality and heritability. Code that is built by associating to each vertex its parent (naïve code), naturally satisfies these properties, but it does not define a bijection between trees and strings.

In this paper we have presented a general scheme for defining bijective codes based on the transformation of a tree into a functional digraph through a function φ. We have emphasized that the required properties are satisfied by our general scheme to the extent that function φ preserves the tree.

The value of our approach is that it returns the characteristics of naïve code as much as is possible: the degree to which function φ preserves the topology of the tree is precisely the same as the degree of similarity between the string obtained using function φ and the string obtained using naïve code. Hence, the required properties are satisfied to this same degree.

We have defined three functions φ_b, φ_m, and φ_d that allow us to fit into our general scheme three known codes: Blob, Happy, and Dandelion codes.

For each of these codes we have shown that it is possible to code and decode in linear time, achieving maximum efficiency. Regarding locality and heritability, we have shown that Happy and Dandelion codes have a performance which is better than Blob code, since they generate functional digraphs with a topology that is very similar to original trees.

Since these codes seem to be suitable candidates for use in evolutionary algorithms, it will be interesting to verify their performance experimentally in tests similar to those reported in [6, 7].

Another interesting view point on these algorithms could be their implementation in a parallel setting: does an efficient parallel way to code and decode trees with high locality and heritability exists?

References

1. Cayley, A.: A theorem on trees. *Quarterly Journal of Mathematics*, 23, pp. 376–378, 1889.

2. Caminiti, S., Finocchi, I., and Petreschi, R.: A unified approach to coding labeled trees. *Proceedings of the 6th Latin American Symposium on Theoretical Informatics*, LNCS 2976, pp. 339–348, 2004. Accepted for publication on Theoretical Computer Science LATIN 2004 Special Issue.
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C.: *Introduction to algorithms*. McGraw-Hill, 2001.
4. Deo, N. and Micikevicius, P.: Parallel algorithms for computing Prüfer-like codes of labeled trees. *Computer Science Technical Report*, CS-TR-01-06, 2001.
5. Edelson, W. and Gargano, M.L.: Feasible encodings for GA solutions of constrained minimal spanning tree problems. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000)*, Morgan Kaufmann Publishers, pp. 754, 2000.
6. Gottlieb, J., Raidl, G., Julstrom, B.A., and Rothlauf F.: Prüfer Numbers: A Poor Representation of Spanning Trees for Evolutionary Search. *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pp. 343-350, 2001.
7. Julstrom, B.A.: The Blob Code: A Better String Coding of Spanning Trees for Evolutionary Search. *In 2001 Genetic and Evolutionary Computation Conference Workshop Program*, pp. 256–261, 2001.
8. Kelmans, A., Pak, I., and Postnikov, A.: Tree and forest volumes of graphs. *DIMACS Technical Report* 2000-03, 2000.
9. Kumar, V., Deo, N., and Kumar, N.: Parallel generation of random trees and connected graphs. *Congressus Numerantium*, 130, pp. 7–18, 1998.
10. Picciotto, S.: *How to encode a tree.* Ph.D. Thesis, University of California, San Diego, 1999.
11. Prüfer, H.: Neuer Beweis eines Satzes über Permutationen. *Archiv für Mathematik und Physik*, 27, pp. 142–144, 1918.
12. Zhou, G. and Gen, M.: A note on genetic algorithms for degree-constrained spanning tree problems. *Networks*, 30(2), pp. 91–95, 1997.

Finding Longest Increasing and Common Subsequences in Streaming Data*

David Liben-Nowell[1], Erik Vee[2,**], and An Zhu[3]

[1] Department of Mathematics and Computer Science, Carleton College
dlibenno@carleton.edu
[2] IBM Almaden Research Center
vee@almaden.ibm.com
[3] Google, Inc.
anzhu@google.com

Abstract. We present algorithms and lower bounds for the Longest Increasing Subsequence (LIS) and Longest Common Subsequence (LCS) problems in the data-streaming model. To decide if the LIS of a given stream of elements drawn from an alphabet Σ has length at least k, we discuss a one-pass algorithm using $O(k \log |\Sigma|)$ space, with update time either $O(\log k)$ or $O(\log \log |\Sigma|)$; for $|\Sigma| = O(1)$, we can achieve $O(\log k)$ space and constant-time updates. We also prove a lower bound of $\Omega(k)$ on the space requirement for this problem for general alphabets Σ, even when the input stream is a permutation of Σ. For finding the actual LIS, we give a $\lceil \log(1+1/\varepsilon) \rceil$-pass algorithm using $O(k^{1+\varepsilon} \log |\Sigma|)$ space, for any $\varepsilon > 0$. For LCS, there is a trivial $\Theta(1)$-approximate $O(\log n)$-space streaming algorithm when $|\Sigma| = O(1)$. For general alphabet Σ, the problem is much harder. We prove several lower bounds on the LCS problem, of which the strongest is the following: it is necessary to use $\Omega(n/\rho^2)$ space to approximate the LCS of two n-element streams to within a factor of ρ, even if the streams are permutations of each other.

1 Introduction

Let $\mathcal{S} = x_1, x_2, \ldots, x_n$ be a sequence of integers. A *subsequence* of $\mathcal{S}$ is a sequence $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ with $i_1 < i_2 < \cdots < i_k$. A subsequence is *increasing* if $x_{i_1} \leq x_{i_2} \leq \cdots \leq x_{i_k}$. We consider two problems: *longest increasing subsequence* – find a maximum-length increasing subsequence of $\mathcal{S}$ – and *longest common subsequence* – given sequences $\mathcal{S}$ and $\mathcal{T}$, find a maximum-length sequence x which is a subsequence of both $\mathcal{S}$ and $\mathcal{T}$. Both LIS and LCS are fundamental combinatorial questions which have been well-studied within computer science.

In the past few years, as we have witnessed the proliferation of truly massive data sets, traditional notions of efficiency have begun to appear inadequate. The theoretical computer science community has thus begun to explore new models

* Part of this work was done while the authors were visiting IBM Almaden. Thanks to D. Sivakumar for suggesting the problem and for fruitful discussions. Thanks also to Graham Cormode, Erik Demaine, Matt Lepinski, and Abhi Shelat.
** Supported in part by NSF grant CCR-0098066.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 263–272, 2005.

of computation, with new notions of efficiency, that more realistically capture when an algorithm is "fast enough." The *data-streaming model* [1] is one such well-studied model. In this model, an algorithm must make a small number of passes over the input data, processing each input element as it goes by. Once the algorithm has seen an element, it is gone forever; thus we must compute and store a small amount of useful information about the previously read input. We are interested in algorithms that use a sublinear (typically polylogarithmic) amount of additional space, with polylogarithmic processing time per element.

In this paper, we study LIS and LCS in the data-streaming model. LIS and LCS are both fundamental combinatorial questions that arise naturally in the streaming context, and they are essentially different from other problems previously studied in this model. We believe that a solid characterization of the tractability of basic questions like LIS and LCS will lead to a greater understanding of the power and limitations of the data-streaming model. In many natural settings we face massive data upon which we must perform LIS and LCS computations – e.g., the optimal-alignment problem in computational biology requires the computation of the LCS of two genomes, and sequentially streaming the data from disk, using a small number of passes over the data, is highly desirable. Another potential use for LIS/LCS in the streaming model is that in certain real-life settings, high-speed data are passing by a bounded-memory device – e.g., a stream of packets passing a router – and we wish to perform some sort of computation on the stream. The question of "what is different about this data stream now, as compared to yesterday?" has been studied from the perspective of large changes in the frequencies of particular elements in the stream [2]; LCS looks at the same question from the perspective of changes in the order of elements in the stream. In a related application, Banerjee and Ghosh [3] have explored the use of LCS as a mechanism for clustering the users of a website on the basis of their "clickstreams" through the site.

One notable difference between LIS (and, similarly, LCS) and problems previously considered in the streaming model is that the LIS of a stream is an essentially global *order-based* property. Many of the problems that have been considered in the streaming model – for example, finding the most frequently occurring items in a stream [4, 5], clustering streaming data [6], finding order statistics for a given stream [7, 8], or calculating the distance between two vectors (presented as a stream of ordered pairs $\langle x_i, i\rangle$) [8–13] – are entirely independent of the order in which the elements are presented in $\mathcal{S}$. Two exceptions are (1) counting *inversions* [14] – i.e., the number of pairs of indices $\langle i, j\rangle$ such that $i < j$ but $x_i > x_j$, and (2) computing a stream's *histogram* [15, 16] – i.e., a compact approximation of the stream by a piecewise constant function. However, there are some significant distinctions between these problems and LIS. Counting inversions is a much more local problem than LIS, in the sense that an inversion is a relation between exactly two items in the stream, whereas an increasing subsequence of length ℓ is a relation among ℓ items. Histograms are much more robust than LIS to small changes in the data: if we consider an LIS that consists primarily of the same repeated value, and we perturb the input so that many occurrences of this value are slightly smaller, then the LIS radically

changes. While these differences do not preclude efficient streaming algorithms for LIS or LCS, they do suggest some of the difficulties.

In this paper, we give a full characterization (up to logarithmic factors) of LCS in the data-streaming model, even in the context of approximation. We also fully characterize (again up to a logarithmic factor) the exact version of LIS, leaving approximations for future work. We first present positive results on LIS. Fredman's algorithm [17] yields a one-pass streaming algorithm that uses $O(k \log |\Sigma|)$ space with update time $O(\log k)$ to compute the length of the LIS for a given input stream, where the elements of the stream are drawn from the (ordered) alphabet $\Sigma = \{1, \ldots, |\Sigma|\}$, and k is the length of the LIS. This algorithm can also achieve an update time of $O(\log \log |\Sigma|)$ [18, 19]. For the problem of *returning* the length-k LIS of a given stream, Fredman's approach gives a one-pass streaming algorithm that uses $O(k^2 \log |\Sigma|)$ space. We reduce the space requirement to $O(k^{1+\varepsilon} \log |\Sigma|)$ by using $\lceil \log(1 + 1/\varepsilon) \rceil$ passes over the data, for any $\varepsilon > 0$. This space usage is nearly optimal, because simply storing the LIS itself requires $\Omega(k \log |\Sigma|)$ space when $|\Sigma| = \Omega(n)$. For streams of elements drawn from a small alphabet, we can achieve $O(|\Sigma| \log k)$ space and $O(\log |\Sigma|)$ update time – i.e., logarithmic space and $O(1)$-time updates if $|\Sigma| = O(1)$ – for computing the length of the LIS. For finding the LIS itself, we achieve the same bounds using $O(|\Sigma|^2 \log k)$ space.

We also present lower bounds on LIS/LCS in the streaming model. (In the comparison model, Fredman has proven lower bounds on computing LIS, via a reduction from sorting [17].) As with many lower bounds in the streaming model, our results are based upon the well-observed connection between the space required by a streaming algorithm and communication complexity. Specifically, a space-efficient streaming algorithm $\mathcal{A}$ to solve a problem gives rise to a solution to the corresponding two-party problem with low communication complexity: one party runs $\mathcal{A}$ on the first part of the input, transmits the small state of the algorithm to the other party, who then continues to run $\mathcal{A}$ on the remainder of the input. We prove a lower bound of $\Omega(k)$ for computing the LIS of a stream whenever $n = \Omega(k^2)$, by giving a reduction from the Set-Disjointness problem, which is known to have high communication complexity. For LCS, we discuss a simple LIS-based algorithm requiring $O(n \log |\Sigma|)$ space to compute the LCS of two n-element sequences presented as streams. We can also approximate LCS in small space: we can achieve a $|\Sigma|$-approximation using $O(|\Sigma| \log n)$ space and $O(1)$ update time. Our main results on LCS, however, are lower bounds. We prove that, if the two streams are general sequences, then we need $\Omega(n)$ space to ρ-approximate the LCS of two streams of length n to within any factor ρ. If the given streams are n-element permutations, we prove that $\Omega(n/\rho^2)$ space is required to ρ-approximate the LCS.

2 Algorithms for Longest Increasing Subsequence

We begin by presenting positive results on LIS, both for computing the length of an LIS and for actually producing an LIS itself. An algorithm to calculate

the length of the LIS was given by Fredman [17] (and later rediscovered by Bespamyatnikh and Segal [20]) in a context other than the data-streaming model. In the data-streaming context, this algorithm yields a one-pass algorithm to find the length k of the LIS using $O(k \log |\Sigma|)$ space and update time of $O(\log k)$ or $O(\log \log |\Sigma|)$. The algorithm also naturally gives a $O(k^2 \log |\Sigma|)$-space algorithm to find the LIS itself. Here we briefly describe Fredman's algorithm, then we give a modification of this algorithm to handle the case of $|\Sigma| = O(1)$, and finally we extend this approach to a more space-efficient multipass streaming algorithm to compute an LIS itself.

Let $\mathcal{S} = x_1, x_2, \ldots, x_n$ be a stream of data, and consider a length-ℓ increasing subsequence $\sigma = x_{i_1}, x_{i_2}, \ldots, x_{i_\ell}$ of $\mathcal{S}$. Write $\mathsf{last}(\sigma) := x_{i_\ell}$. Let σ_i denote the ith element in a subsequence σ. For instance, $\mathsf{last}(\sigma) := \sigma_{|\sigma|}$. We say that σ is $\langle \ell, j \rangle$-*minimal* if $\mathsf{last}(\sigma)$ is minimized over all length-ℓ increasing subsequences of the substream $x_1, x_2, \ldots, x_j$. We say that such a sequence σ is an $\langle \ell, j \rangle$-*minimal increasing sequence*, or simply an $\langle \ell, j \rangle$-*MIS*. The streaming algorithm of Fredman [17] works as follows: we maintain an array $A[1 \ldots k']$, where, after we have scanned the first j elements of the stream, $A[\ell]$ stores $\mathsf{last}(\sigma)$ for an $\langle \ell, j \rangle$-MIS σ. We update A with x_j by setting $A[\ell + 1] := x_j$ if $A[\ell] \leq x_j < A[\ell + 1]$. Every step of the algorithm takes $O(1)$ time except for identifying ℓ, which can be done in $O(\log k)$ time by binary search; alternatively, we can use van Emde Boas queues [18] or y-fast trees [19] to support updates in $O(\log \log |\Sigma|)$ time.

When the alphabet is of small size, we can do much better. We maintain an array $B[1 \ldots |\Sigma|]$ such that $A[B[j]] = j < A[B[j] + 1]$, for each $j \in \Sigma$. That is, after $x_1, \ldots, x_i$ have been read, the quantity $B[j]$ denotes the length ℓ of the longest $\langle \ell, i \rangle$-MIS that ends with the element j. (In the case that $A[\ell] \neq j$ for all ℓ, we keep $B[j]$ uninitialized.) When a new element x_i of the stream arrives, we simply reset $B[x_i] := 1 + \max\{B[1], \ldots, B[x_i - 1], B[x_i]\}$. The length of the LIS is then $\max\{B[1], \ldots, B[|\Sigma|]\}$. Notice that although we refer to the array A conceptually, there is no need to actually maintain it in an implementation. Hence, we can compute the length of the LIS exactly using $O(|\Sigma| \log k)$ space, with $O(|\Sigma|)$ update time. The update time can be improved to $O(\log |\Sigma|)$ by placing a complete binary tree on top of the array B, with each node augmented to store the maximum value beneath it in the tree. (This data structure is a simplified version of order-statistic trees [21].) When $|\Sigma|$ is constant, this requirement is $O(\log k)$ space and $O(1)$ update time. We thus have the following.

Theorem 1. *We can decide whether the LIS of a stream of integers from Σ has length at least a given number k, or compute the length k of the LIS of the given stream, with a one-pass streaming algorithm that uses $O(k \log |\Sigma|)$ space and has update time $O(\log k)$ or $O(\log \log |\Sigma|)$. Alternatively, we may compute the length k of the LIS of the given stream with a one-pass streaming algorithm that uses $O(|\Sigma| \log k)$ space and has update time $O(\log |\Sigma|)$. For $|\Sigma|$ constant, this requirement is $O(\log k)$ space and $O(1)$ update time.* □

To return the actual LIS of a stream, we modify the above algorithm to maintain, for each ℓ, a length-ℓ sequence σ^ℓ whose last element is $A[\ell]$. When we update $A[\ell + 1] := x_i$, we reset $\sigma^{\ell+1} := \sigma^\ell, x_i$. This modification adds only a constant

amount of extra running time per update, so the update time per element remains $O(\log k)$ or $O(\log\log|\Sigma|)$, and the space requirement is $O(k^2\log|\Sigma|)$.

We now describe a p-pass algorithm that requires less space. The key modification is that during the first pass over the data, the algorithm only remembers part of each σ^ℓ, specifically every qth element (for q specified below). For each ℓ, we maintain the sequence $\widetilde{\sigma}^\ell = \sigma^\ell_1, \sigma^\ell_{q+1}, \sigma^\ell_{2q+1}, \ldots, \sigma^\ell_{\lfloor(\ell-2)/q\rfloor q+1}, \sigma^\ell_\ell$, where $\sigma^\ell_\ell = A[\ell]$, as before. After the first pass, we discard the subsequences $\widetilde{\sigma}^1, \ldots, \widetilde{\sigma}^{k-1}$, retaining only $\widetilde{\sigma}^k = \sigma^k_1, \sigma^k_{q+1}, \sigma^k_{2q+1}, \ldots, \sigma^k_{\lfloor(k-2)/q\rfloor q+1}, \sigma^k_k$ where σ^k is a length-k LIS of the input. We may then fill in the gaps in a recursive fashion on subsequent passes. Specifically, the algorithm uses $p-1$ passes to find the length-q increasing subsequence starting with the element $\widetilde{\sigma}^k_i$ and ending at $\widetilde{\sigma}^k_{i+1}$ for each $i = 1, 2, \ldots, k-1$. Let $S(k,p)$ denote the space required by a p-pass algorithm to find a subsequence of length k. We then have the following recurrence: $S(k,p) = \max(O(k^2\log|\Sigma|/q), S(q,p-1)) + O(k\log|\Sigma|)$. Solving the recurrence, we find that the space requirements are optimized when $q = k^{1-1/(2^p-1)}$, and where $S(k,p) = O(k^{1+1/(2^p-1)}\log|\Sigma|)$.

Theorem 2. *Fix $\varepsilon > 0$. For a given k, we can find a length-k increasing subsequence of a stream of integers from Σ with a $\lceil\log(1+1/\varepsilon)\rceil$-pass streaming algorithm using $O(k^{1+\varepsilon}\log|\Sigma|)$ space and has update time $O(\log k)$ or $O(\log\log|\Sigma|)$. We can find the LIS even when its length k is not known in advance, using the same number of passes and update time, and space $O(\frac{1}{\varepsilon}k^{1+\varepsilon}\log|\Sigma|)$.* □

Actually finding an LIS is also easier for small $|\Sigma|$. Rather than maintaining σ^ℓ for each ℓ, we need only maintain $\sigma^{B[j]}$ for each $j \in \Sigma$. Further, we only need $O(|\Sigma|\log k)$ space for each sequence σ^ℓ, because it suffices to keep track of the indices i for which $\sigma^\ell_i \neq \sigma^\ell_{i+1}$, and there are at most $|\Sigma|$ such indices. Hence, we can find the LIS using space $O(|\Sigma|^2\log k)$ and update time $O(\log|\Sigma|)$. When $|\Sigma| = O(1)$, this requirement is just $O(\log k)$ space and $O(1)$ update time.

3 Lower Bounds for LIS

We now turn to a lower bound on the space requirements for streaming algorithms for LIS. We prove that $\Omega(k)$ bits of storage are required to decide if the LIS of a stream of N elements has length at least k, for any $N = \Omega(k^2)$. Our proof is based on a reduction from the *set-disjointness problem*:

Definition 3 (Set Disjointness). *Party A holds an n-bit string s_A, and Party B holds another n-bit string s_B. The pair $\langle s_A, s_B\rangle$ is a 'yes' instance for Set Disjointness iff the ith bit of both s_A and s_B is 1, for some i. The* communication complexity *for a protocol solving* Set-Disjointness *is the maximum number of bits communicated between Party A and Party B, taken over all valid inputs.*

Lower bounds for the set-disjointness problem have been studied extensively (e.g., [9, 22, 23]). The most recent results show that even in the randomized setting, Set-Disjointness requires a large amount of communication:

Theorem 4 ([9]). *Let $\delta \in (0, 1/4)$. Any randomized protocol solving the* Set-Disjointness *problem with probability at least $1-\delta$ requires at least $\frac{n}{4}(1-2\sqrt{\delta})$ bits of communication, even when s_A and s_B both contain exactly $n/4$ ones.* □

We now reduce Set-Disjointness to the problem of determining if an increasing subsequence of length $\sqrt{N}$ exists in an N-element stream. This reduction shows that – even with randomization and some chance of error – deciding whether the LIS has length k requires $\Omega(k)$ space in the streaming model.

Suppose we are given an instance $\langle s_A, s_B \rangle$ of Set-Disjointness, where $n := |s_A| = |s_B|$. We construct a stream $\mathsf{S\text{-}lis}(s_A, s_B)$ whose first half depends only on s_A, whose second half depends only on s_B, and whose LIS has length at least $n+1$ if and only if $\langle s_A, s_B \rangle$ are non-disjoint. For each index $i \in \{1, \ldots, n\}$, let $N_i = (n+1) \cdot (i-1)$ for readability. We define $\mathsf{A\text{-}part}(i)$ to be the length-i increasing sequence $N_i + 1, N_i + 2, \ldots, N_i + i$ and $\mathsf{B\text{-}part}(i)$ to be the length-$(n-i+1)$ increasing sequence $N_i+i+1, N_i+i+2, \ldots, N_{i+1}$. Let $\mathsf{S\text{-}lis}_A(s_A)$ be the sequence consisting of the concatenation of the sequences $\mathsf{A\text{-}part}(i)$ for every $i \in \{i : s_A(i) = 1\}$, listed in decreasing order of the index i. Similarly, let $\mathsf{S\text{-}lis}_B(s_B)$ be the sequence consisting of the sequences $\mathsf{B\text{-}part}(i)$ for every $i \in \{i : s_B(i) = 1\}$, also listed in decreasing order of the index i. Clearly, $\mathsf{S\text{-}lis}_A(s_A)$ (and $\mathsf{S\text{-}lis}_B(s_B)$, respectively) only depends on s_A (and s_B, respectively). Then we define the stream $\mathsf{S\text{-}lis}(s_A, s_B)$ to be $\mathsf{S\text{-}lis}_A(s_A)$ followed by $\mathsf{S\text{-}lis}_B(s_B)$.

Lemma 5. *If vectors s_A, s_B intersect, then $|\mathsf{LIS}(\mathsf{S\text{-}lis}(s_A, s_B))| \geq n+1$. If vectors s_A, s_B do not intersect, then $|\mathsf{LIS}(\mathsf{S\text{-}lis}(s_A, s_B))| \leq n$.*

Proof. If $s_A(i) = s_B(i) = 1$, then $\mathsf{S\text{-}lis}(s_A, s_B)$ contains the length-$(n+1)$ increasing subsequence $\mathsf{A\text{-}part}(i), \mathsf{B\text{-}part}(i)$. Conversely, suppose s_A and s_B do not intersect. Whenever $i < j$, we have that (1) $\mathsf{A\text{-}part}(i)$ follows $\mathsf{A\text{-}part}(j)$ in $\mathsf{S\text{-}lis}_A(s_A)$ and (2) the integers in $\mathsf{A\text{-}part}(i)$ are all smaller than those in $\mathsf{A\text{-}part}(j)$. Thus the LIS within $\mathsf{S\text{-}lis}_A(s_A)$ has length at most n and can contain integers from $\mathsf{A\text{-}part}(i)$ for only a single i, and analogously for $\mathsf{S\text{-}lis}_B(s_B)$. Thus the only potential LIS of length $n+1$ must be subsequences of $\mathsf{A\text{-}part}(i), \mathsf{B\text{-}part}(j)$ for some indices i and j so that $s_A(i) = s_B(j) = 1$. (Thus $i \neq j$.) Furthermore, unless $i < j$, all the integers in $\mathsf{A\text{-}part}(i)$ are larger than the integers in $\mathsf{B\text{-}part}(j)$. Thus the LIS in $\mathsf{S\text{-}lis}(s_A, s_B)$ is of length at most $|\mathsf{A\text{-}part}(i)| + |\mathsf{B\text{-}part}(j)| = i+n-j+1 \leq n$. □

We now improve the construction so that the resulting stream $\mathsf{S\text{-}lis}(s_A, s_B)$ is a *permutation*, i.e., contains each element of $\{0, 1, \ldots, \ell\}$ exactly once. We will show that a suitable $\ell = \Theta(n^2)$ suffices. We modify $\mathsf{S\text{-}lis}_A$ and $\mathsf{S\text{-}lis}_B$ as follows: we include the integers from $\mathsf{A\text{-}part}(i)$ and $\mathsf{B\text{-}part}(i)$ even when $s_A(i) = 0$ or $s_B(i) = 0$, but so that only two of these elements can be part of an LIS:

- Let $U_A := \{x \mid \exists i : s_A(i) = 0, x \in \mathsf{A\text{-}part}(i)\}$. Then we define $\mathsf{pad\text{-}A}(s_A)$ to be the sequence consisting of integers in U_A listed in decreasing order, followed by 0. We define $\mathsf{S\text{-}lis}^{\pi}_A(S_A)$ to be $\mathsf{pad\text{-}A}(s_A)$ followed by $\mathsf{S\text{-}lis}_A(s_A)$.
- Let $U_B := \{x \mid \exists i : s_B(i) = 0, x \in \mathsf{B\text{-}part}(i)\}$. Define $\mathsf{pad\text{-}B}(s_B)$ to be the sequence consisting of $(n+1) \cdot n + 1$, followed by the integers in U_B listed in decreasing order. Define $\mathsf{S\text{-}lis}^{\pi}_B(S_B)$ as $\mathsf{S\text{-}lis}_B(s_B)$ followed by $\mathsf{pad\text{-}B}(s_B)$.

Now define $\mathsf{S\text{-}lis}^{\pi}(s_A, s_B) := \mathsf{S\text{-}lis}^{\pi}_A(s_A), \mathsf{S\text{-}lis}^{\pi}_B(s_B)$. One can easily verify that $\mathsf{S\text{-}lis}^{\pi}(s_A, s_B)$ is a permutation of the set $\{0, \ldots, (n+1)\cdot n+1\}$, and that $\mathsf{pad\text{-}A}(s_A)$ and $\mathsf{pad\text{-}B}(s_B)$ each increase the LIS by exactly one.

Lemma 6. *If vectors s_A, s_B intersect, then $|\mathsf{LIS}(\mathsf{S\text{-}lis}^{\pi}(s_A, s_B))| \geq n + 3$. If vectors s_A, s_B do not intersect, then $|\mathsf{LIS}(\mathsf{S\text{-}lis}^{\pi}(s_A, s_B))| \leq n + 2$.* □

Theorem 7. *For any length k and for any $N \geq k(k-1)+2$, any streaming algorithm which decides whether $\mathsf{LIS}(\mathcal{S}) \geq k$ for a stream $\mathcal{S}$ that is a permutation of $\{1, \ldots, N\}$ with probability at least $3/4$ requires $\Omega(k)$ space.*

Proof. Suppose that an algorithm $\mathcal{A}(S)$ decides with probability at least $3/4$ whether stream S contains an increasing subsequence of length k, where $|\mathcal{S}| = N$. We show how to solve an instance $\langle s_A, s_B \rangle$ of the Set-Disjointness problem with $|s_A| = k - 1 = |s_B|$ with probability at least $3/4$ by calling $\mathcal{A}$. The stream we consider is $\mathcal{S} := \mathsf{Extra\_Numbers}, \mathsf{S\text{-}lis}^{\pi}(s_A, s_B)$, where $\mathsf{Extra\_Numbers} := N - 1, N-2, \ldots, k\cdot(k-1)+2$. Then the LIS of $\mathcal{S}$ has exactly the same length as the LIS of $\mathsf{S\text{-}lis}^{\pi}(s_A, s_B)$ because the prepended elements of $\mathcal{S}$ are all larger than those in $\mathsf{S\text{-}lis}^{\pi}(s_A, s_B)$, and they are presented in descending order. Thus, by Lemma 6, the LIS of $\mathcal{S}$ has length k – and $\mathcal{A}(\mathcal{S})$ returns true with probability at least $3/4$ – if and only if s_A and s_B do not intersect. A lower bound on the space required by $\mathcal{A}$ follows: to solve the instance $\langle s_A, s_B \rangle$ of the Set-Disjointness problem, Party A simulates the algorithm $\mathcal{A}$ on the stream $\mathsf{Extra\ Numbers}, \mathsf{S\text{-}lis}^{\pi}_A(s_A)$, then sends all stored information to Party B, who continues simulating $\mathcal{A}$ on the remainder of the stream $\mathcal{S}$. By Theorem 4, then, Party A must transmit at least $\Omega(k)$ bits in this protocol, and thus $\mathcal{A}$ must use $\Omega(k)$ space. □

4 Longest Common Subsequence

We now turn to the LCS problem. We consider the *adversarial* streaming model, where elements from the two streams can be presented in any order of interleaving. In our lower bounds, the algorithm is presented all of $\mathcal{S}_1$ before any of $\mathcal{S}_2$.

As with all streaming problems, there is a trivial streaming algorithm using $\Theta(n \log |\Sigma|)$ space: simply store both n-element streams in their entirety and then run a standard LCS algorithm. There is also a trivial $|\Sigma|$-approximation working in $O(|\Sigma| \log n)$ space and $O(1)$ update time. For each element $a \in \Sigma$, calculate the value of k_a so that a appears at least k_a times in both $\mathcal{S}_1$ and $\mathcal{S}_2$. The maximum k_a is within a factor of $|\Sigma|$ of the optimal LCS. When $|\Sigma| = O(1)$, we then have a constant-factor approximation streaming algorithm to LCS using logarithmic space. We can give another algorithmic upper bound for a version of LCS, based upon a simple connection with LIS. Suppose that we are first given one *reference permutation* $\mathcal{R}$ and then given a large number of *test permutations* $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_q$; we want to compute the LCS of $\mathcal{R}$ and $\mathcal{S}_i$. Our streaming algorithm stores the permutation $\mathcal{R}$ as a lookup table and then, for each $\mathcal{S}_i$, runs the LIS algorithm from Section 2, where we take $x < y$ if x appears before y in $\mathcal{R}$. This algorithm requires $O(n \log |\Sigma|)$ total space – $O(n \log |\Sigma|)$ to

store $\mathcal{R}$, and $O(k \log |\Sigma|) = O(n \log |\Sigma|)$ for the LIS computation. Note that this bound is independent of q.

Lower Bounds for Exact/Approximate LCS for Non-permutations. It is straightforward to see that if we do not require the streams $\mathcal{S}_1$ and $\mathcal{S}_2$ to be permutations of each other, then the lower bound is trivial, even for approximation:

Theorem 8. *For any length $N \geq 1$ and approximation ratio $\rho > 1$, any streaming algorithm which ρ-approximates the LCS of two streams $\mathcal{S}_1, \mathcal{S}_2$ (in adversarial order) each of length N with probability at least $3/4$ requires $\Omega(N)$ space.*

Proof. Let $\mathcal{S}$ be a sequence consisting of sequence $\mathcal{S}_1$ followed by sequence $\mathcal{S}_2$. Suppose that an algorithm $\mathcal{A}(\mathcal{S})$ decides with probability at least $3/4$ whether streams $\mathcal{S}_1$ and $\mathcal{S}_2$ contain an LCS of length one. We show how to solve an instance $\langle s_A, s_B \rangle$ of Set-Disjointness with probability at least $3/4$ with $|s_A| = 4N = |s_B|$, where s_A and s_B both contain exactly N ones, by using $\mathcal{A}$.

Let stream $\mathcal{S}_1$ consist of all indices i such that $s_A(i) = 1$, and let $\mathcal{S}_2$ consist of all indices i such that $s_B(i) = 1$. Thus $\mathsf{LCS}(\mathcal{S}_1, \mathcal{S}_2) \geq 1$ if s_A and s_B have at least one element in common and $\mathsf{LCS}(\mathcal{S}_1, \mathcal{S}_2) = 0$ otherwise. Thus, if $\mathcal{A}(\mathcal{S})$ outputs the correct answer within any ratio ρ, it must distinguish between the length-0 and length-1 cases. This fact implies the desired lower bound, because we can solve the Set-Disjointness using $\mathcal{A}$ as in Theorem 7. To show that we still require $\Omega(N)$ space when one or both of the streams has length strictly larger than N, we simply add arbitrary new elements to each of the above streams. □

Although this construction is for multiplicative approximation, a simple variant shows that any data-streaming algorithm solving LCS within additive α takes space at least $\Omega(N/\alpha)$; simply repeat each element in the streams $2\alpha + 1$ times.

Lower Bounds for Exact LCS for Permutations. We now improve the construction to show a lower bound on the space required for LCS even when $\mathcal{S}_1$ and $\mathcal{S}_2$ are both permutations of the set $\{1, \ldots, n\}$. Given an instance $\langle s_A, s_B \rangle$ of the Set-Disjointness problem where there are exactly $n/4$ ones in each s_A and s_B, we construct two streams as follows: (1) $\mathsf{S\text{-}lcs}^{\pi}_A(s_A) = R_A, \overline{R}_A$, where R_A contains $\{i : s_A(i) = 1\}$ in increasing order and $\overline{R}_A$ contains $\{i : s_A(i) = 0\}$ in decreasing order; and (2) $\mathsf{S\text{-}lcs}^{\pi}_B(s_B) = R_B, \overline{R}_B$, where R_B contains $\{i : s_B(i) = 1\}$ in increasing order and $\overline{R}_B$ contains $\{i : s_B(i) = 0\}$ in decreasing order.

Lemma 9. *If vectors s_A and s_B intersect then $|\mathsf{LCS}(\mathsf{S\text{-}lcs}^{\pi}_A(s_A), \mathsf{S\text{-}lcs}^{\pi}_B(s_B))| \geq n/2 + 2$. Otherwise, the length of the LCS is at most $n/2 + 1$.* □

Theorem 10. *For any length k and for any $N \geq 2k - 4$, any streaming algorithm which decides whether $\mathsf{LCS}(\mathcal{S}_1, \mathcal{S}_2) \geq k$ for streams $\mathcal{S}_1, \mathcal{S}_2$ which are permutations of $\{1, \ldots, N\}$ with probability at least $3/4$ requires $\Omega(k)$ space.*

Proof. Analogous to Theorem 8 if $N = 2k - 4$. For larger N, we pad the streams, as in Theorem 7, by adding the decreasing sequence $N, N-1, N-2, \ldots, 2k-4+1$ to the beginning of $\mathsf{S\text{-}lcs}^{\pi}_A(s_A)$ and the increasing sequence $2k - 4 + 1, 2k - 4 + 2, \ldots, N$ to the end of $\mathsf{S\text{-}lcs}^{\pi}_B(s_B)$. As before, the LCS has length k if and only if s_A and s_B intersect, and thus we require $\Omega(k)$ space to compute the LCS. □

Lower Bounds for Approximate LCS for Permutations. Suppose we wish to ρ-approximate LCS on permutations for a desired factor $\rho > 1$. For each i, we construct sequences $\mathsf{apx}_\rho\mathsf{A}(i,s_A), \mathsf{apx}_\rho\mathsf{B}(i,s_B)$ so $|\mathsf{LCS}(\mathsf{apx}_\rho\mathsf{A}(i,s_A), \mathsf{apx}_\rho\mathsf{B}(i,s_B))|$ is ρ^2 if $s_A(i) = s_B(i) = 1$ and is at most ρ otherwise. For each $i \leq n$, both sequences are of length ρ^2, and consist of all of the integers $Z^{i,\rho} := \{(i-1)\cdot\rho^2 + 1, (i-1)\cdot\rho^2 + 2, \ldots, (i-1)\cdot\rho^2 + \rho^2\}$.

- Let $\mathsf{apx}_\rho\mathsf{A}(i,s_A)$ be $Z^{i,\rho}$ presented in increasing order if $s_A(i) = 1$, and $Z^{i,\rho}$ presented in decreasing order if $s_A(i) = 0$.
- For $s_B(i) = 1$, let $\mathsf{apx}_\rho\mathsf{B}(i,s_B)$ to be $Z^{i,\rho}$ presented in increasing order. For $s_B(i) = 0$, we use the more complicated *median ordering* of $Z^{i,\rho}$, so that the LIS and the longest decreasing subsequence both have length exactly ρ. (For the set $\{1,\ldots,m^2\}$, this is achieved by the sequence $m, m-1, \ldots, 1; 2m, 2m-1, \ldots, m+1; \ldots; m^2, m^2-1, \ldots, m(m-1)+1$.)

Given $\langle s_A, s_B\rangle$ with exactly $n/4$ ones in s_A, s_B, we construct two streams:

$$\mathsf{S\text{-}apx}_\rho\text{-}\mathsf{lcs}^\pi_A(s_A) := \mathsf{apx}_\rho\mathsf{A}(1,s_A), \mathsf{apx}_\rho\mathsf{A}(2,s_A), \ldots, \mathsf{apx}_\rho\mathsf{A}(n,s_A)$$
$$\mathsf{S\text{-}apx}_\rho\text{-}\mathsf{lcs}^\pi_B(s_B) := \mathsf{apx}_\rho\mathsf{B}(n,s_B), \mathsf{apx}_\rho\mathsf{B}(n-1,s_B), \ldots, \mathsf{apx}_\rho\mathsf{B}(1,s_B).$$

Lemma 11. *If s_A, s_B intersect, then $|\mathsf{LCS}(\mathsf{S\text{-}apx}_\rho\text{-}\mathsf{lcs}^\pi_A(s_A), \mathsf{S\text{-}apx}_\rho\text{-}\mathsf{lcs}^\pi_B(s_B))| \geq \rho^2$. If s_A, s_B do not intersect, then the length of the LCS is at most ρ.* □

Theorem 12. *For any approximation ratio ρ, and for any N, any streaming algorithm which decides whether (i) $\mathsf{LCS}(\mathcal{S}_1, \mathcal{S}_2) \geq \rho^2$ or (ii) $\mathsf{LCS}(\mathcal{S}_1, \mathcal{S}_2) \leq \rho$ for streams $\mathcal{S}_1, \mathcal{S}_2$ which are permutations of $\{1,\ldots,N\}$ with probability at least $3/4$ requires $\Omega(N/\rho^2)$ space (and thus ρ-approximating the LCS of N-element permutations requires $\Omega(N/\rho^2)$ space).* □

5 Conclusion and Future Work

A classic theorem of Erdös and Szekeres [24] follows from an elegant application of the pigeonhole principle: for any sequence $\mathcal{S}$ of $n+1$ numbers, there is either an increasing subsequence of $\mathcal{S}$ of length $\sqrt{n}$ or a decreasing subsequence of $\mathcal{S}$ of length $\sqrt{n}$. One of our original motivations for looking at the LIS problem was to consider the difficulty of deciding, given a stream $\mathcal{S}$, whether (1) the length of the LIS of $\mathcal{S}$ is at least $\sqrt{|S|}$, (2) the length of the longest *decreasing* sequence is at least $\sqrt{|S|}$, or (3) both. To do this, one needs an *exact* streaming algorithm for LIS; a minor modification to the median sequence in Section 4 shows that one can have an LIS of length $\sqrt{n}$ or length $\sqrt{n}-1$ with a longest decreasing subsequence of length $\sqrt{n}$ or length $\sqrt{n}+1$, respectively. Of course, in the streaming model one is usually interested in *approximate* algorithms using, say, polylogarithmic space. Our lower bounds for LCS show that one needs a large amount of space for any reasonable approximation. However, our lower bounds for the LIS problem say that a streaming algorithm that distinguishes between an LIS of length k and one of length $k+1$ requires $\Omega(k)$ space. It is an interesting open question whether one can use a small amount of space to approximate LIS in the streaming model.

References

1. Henzinger, M.R., Raghavan, P., Rajagopalon, S.: Computing on data streams. Technical Report 1998-011, Digital Equipment Corp., Systems Res. Center (1998)
2. Cormode, G., Muthukrishnan, S.: What's new: Finding significant differences in network data streams. In: Proc. INFOCOM. (2004)
3. Banerjee, A., Ghosh, J.: Clickstream clustering using weighted longest common subsequence. In: ICM Workshop on Web Mining. (2001)
4. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Proc. ICALP. (2002)
5. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of Internet packet streams with limited space. In: Proc. ESA. (2002) 348–360
6. Guha, S., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams. In: Proc. FOCS. (2000) 359–366
7. Manku, G., Rajagopalan, S., Lindsay, B.: Approximate medians and other quantiles in one pass and with limited memory. In: Proc. SIGMOD. (1998)
8. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. JCSS **58** (1999) 137–147
9. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. In: Proc. FOCS. (2002)
10. Feigenbaum, J., Kannan, S., Strauss, M., Viswanathan, M.: An approximate L_1-difference algorithm for massive data streams. In: Proc. FOCS. (1999)
11. Fong, J., Strauss, M.: An approximate L_p-difference algorithm for massive data streams. In: Proc. STACS. (2000)
12. Indyk, P.: Stable distributions, pseudorandom generators, embeddings, and data stream computations. In: Proc. FOCS. (2000)
13. Saks, M., Sun, X.: Space lower bounds for distance approximation in the data stream model. In: Proc. STOC. (2002)
14. Ajtai, M., Jayram, T.S., Kumar, R., Sivakumar, D.: Approximate counting of inversions in a data stream. In: Proc. STOC. (2002)
15. Gilbert, A., Guha, S., Indyk, P., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Fast, small-space algorithms for approximate histogram maintenance. In: STOC. (2002)
16. Guha, S., Koudas, N., Shim, K.: Data-streams and histograms. In: STOC. (2001)
17. Fredman, M.L.: On computing the length of longest increasing subsequences. Discrete Mathematics **11** (1975) 29–35
18. van Emde Boas, P.: Preserving order in a forest in less than logarithmic time and linear space. IPL **6** (1977) 80–82
19. Willard, D.E.: Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. IPL **17** (1983) 81–84
20. Bespamyatnikh, S., Segal, M.: Enumerating longest increasing subsequences and patience sorting. IPL **76** (2000) 7–11
21. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms. McGraw-Hill (2002)
22. Kalyanasundaram, B., Schnitger, G.: The probabilistic communication complexity of set intersection. SIAM J. Disc. Math **5** (1992) 545–557
23. Razborov, A.: On the distributional complexity of disjointness. JCSS **28** (1984)
24. Erdös, P., Szekeres, G.: A combinatorial problem in geometry. Compositio Mathematica (1935) 463–470

$O(n^2 \log n)$ Time On-Line Construction of Two-Dimensional Suffix Trees

Joong Chae Na[1], Raffaele Giancarlo[2,⋆], and Kunsoo Park[1,⋆⋆]

[1] School of Computer Science and Engineering, Seoul National University, Korea
{jcna,kpark}@theory.snu.ac.kr
[2] Dipartimento di Matematica ed Applicazioni, Università di Palermo, Italy
raffaele@math.unipa.it

Abstract. The *two-dimensional suffix tree* of an $n \times n$ square matrix A is a compacted trie that represents all *square submatrices* of A [9]. For the off-line case, i.e., A is given in advance to the algorithm, it is known how to build it in optimal time, for any type of alphabet size [9, 15]. Motivated by applications in Image Compression [18], Giancarlo and Guaiana [12] considered the on-line version of the two-dimensional suffix tree and presented an $O(n^2 \log^2 n)$-time algorithm, which we refer to as GG. That algorithm is a non-trivial generalization of Ukkonen's on-line algorithm for standard suffix trees [19]. The main contribution in this paper is an $O(\log n)$ factor improvement in the time complexity of the GG algorithm, making it optimal for unbounded alphabets [7]. Moreover, the ideas presented here also lead to a major simplification of the GG algorithm. Technically, we are able to preserve most of the structure of the original GG algorithm, by reducing a computational bottleneck to a primitive operation, i.e., comparison of Lcharacters, which is here implemented in constant time rather than $O(\log n)$ time as in GG. However, preserving that structure comes at a price. Indeed, in order to make everything work, we need a careful reorganization of another fundamental algorithm: Weiner's algorithm for the construction of standard suffix trees [20]. Specifically, here we provide a version of that algorithm which takes linear time and works *on-line* and *concurrently* over a set of strings.

1 Introduction

The suffix tree is a compacted trie that represents all substrings of a given string. Over the years, it has gained the status of a fundamental data structure not only for Computer Science, but also for Engineering and Bioinformatics applications [2, 6, 13]. It should not be surprising that it has been the object of quite a bit of algorithmic investigation [3, 7, 17, 19, 20].

⋆ Partially supported by the Italian MIUR projects "Bioinformatica per la Genomica e la Proteomica", and "Metodi Combinatori ed Algoritmici per la Scoperta di Patterns in Biosequenze". Additional support provided by a CNRS Fellowship, France, sponsoring visits to Institut Gaspard Monge, Marne La Vallee, France.
⋆⋆ Corresponding Author. Supported by the MOST Grant M6-0203-00-0039.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 273–282, 2005.

In the late 80's-early 90's, many of the basic string matching algorithms and data structures were studied in "higher dimensions", i.e., for matrices. The interested reader may find relevant references as well as an account of the evolution of that subject in [1, 11]. Here we are interested in the two-dimensional analog of the suffix tree, i.e., a data structure that stores all submatrices of an $n \times m$ matrix, $n \geq m$. Unfortunately, building that data structure takes $\Omega(nm^2)$ time [8]. So, attention has been restricted to square matrices and submatrices.

In that context, Giancarlo [9] proposed the *Lsuffix tree*, compactly storing all *square submatrices* of an $n \times n$ matrix A, together with an $O(n^2 \log n)$-time construction algorithm that uses $O(n^2)$ space. Giancarlo and Grossi [10, 11] also introduced the general framework of two-dimensional suffix tree families and gave an expected linear-time construction algorithm. Kim and Park [15], and Cole and Hariharan [5] gave a worst-case linear-time and randomized linear-time construction algorithms, respectively. Two-dimensional suffix arrays (extension of Manber and Myers's suffix arrays [16]) are also constructed in $O(n^2 \log n)$ time [9, 14]. All of the mentioned results apply to the off-line case, i.e., the matrix is given in advance to the algorithm and does not change during its execution. However, Storer [18] showed that the on-line case, i.e., the matrix is given one row or column at a time, would be of considerable theoretical interest for Data Compression. Indeed, he obtained generalizations to matrices, i.e., images, of LZ1-type compression methods that would greatly benefit from efficient algorithms for the on-line construction of the Lsuffix tree. Motivated by the findings by Storer, Giancarlo and Guaiana [12] presented an $O(n^2 \log^2 n)$-time algorithm for *on-line* construction of the Lsuffix Tree. We refer to that algorithm as the *GG algorithm*. It is a nontrivial extension to square matrices of the algorithm by Ukkonen for the on-line construction of the standard suffix tree [19].

Our main contribution is an $O(\log n)$ time improvement in the *GG* algorithm. That yields the first optimal algorithm for the on-line construction of the Lsuffix tree over unbounded alphabets [7]. We assume that an $n \times n$ matrix A is read on-line in column or row major order. Our algorithm keeps most of the structure of the *GG* algorithm. Indeed, based on a careful analysis of that algorithm, we identify a computational bottleneck and formalize it as a primitive operation needed by the algorithm. That operation is the comparison of Lcharacters which, intuitively, are the analog of characters for strings and are therefore considered as atomic items in the *GG* algorithm. Here we are able to provide techniques by means of which the comparison of two Lcharacters requires constant time rather than $O(\log n)$ time as in the *GG* algorithm. We point out that, for off-line Lsuffix tree construction, it is known how to compare two Lcharacters in constant time, since we can preprocess the entire matrix and therefore all the needed auxiliary data structures for that comparison are *static*. In the on-line case, the auxiliary data structures are dynamic and it is not so obvious that one can compare Lcharacters in constant time. Moreover, our implementation of the primitive operation alluded to before must be consistent with the remaining part of the *GG* algorithm. As a by-product of our new technique, we also obtain a simpler version of the *GG* algorithm.

Our main result comes from an apparently novel solution to the following problem, which may be of independent interest:

Problem We are given a set of m strings $x_1, \ldots, x_m$ of the same length n. At time p, $p = 1, \ldots, n$, we only know the first p characters of each string, i.e., $x_i[1..p]$, $1 \leq i \leq m$. We want a data structure that, at time p, takes in input (i_1, j_1) and (i_2, j_2), $1 \leq i_1, i_2 \leq m$ and $1 \leq j_1, j_2 \leq p$, and returns the length of the longest common suffix between $x_{i_1}[1..j_1]$ and $x_{i_2}[1..j_2]$.

We note that for the solution of the problem just stated, it is not evident how to use either McCreight's or Farach-Colton et al.'s suffix tree algorithms because they are not on-line. Use of Ukkonen's on-line algorithm yields an $O(\log n)$ solution per query, as shown in [12]. That is due to the fact that Ukkonen's algorithm does not maintain a one-to-one correspondence between the leaves of the tree and the suffixes of the strings, while the strings are read on-line and the tree is being built. In fact, only at the end there will be such a correspondence. Here we are able to modify Weiner's algorithm so that we can obtain constant time per query and overall linear time. Indeed, we give a version of Weiner's algorithm that works *on-line* and *cuncurrently* for a set of strings. We also make use of recent results by Cole and Hariharan on dynamic LCA queries [4].

2 Preliminaries

2.1 One-Dimensional Suffix Trees

Let S be a string of length n over a fixed alphabet Σ. We denote the ith character by $S[i]$ and the substring $S[i]S[i+1] \cdots S[j]$ by $S[i..j]$. We assume that $S[n] = \#$ is a unique character which is lexicographically smaller than any other character in Σ. For $1 \leq i \leq n$, the *suffix* $S(i)$ of S is the longest substring of S that starts at position i in S. We denote by α^R the reversed string of a string α.

The *suffix tree* T of string S is a compacted trie over the alphabet Σ that represents all suffixes of S. Each internal node, other that the root, has at least two children and each edge is labelled with a nonempty substring of S. No two edges out of a node can have edge-labels beginning with the same character. We refer to [7, 13, 17, 19, 20] for a more formal description as well as linear-time construction algorithms.

In this paper, we make use of Weiner's algorithm [3, 20]. It constructs a suffix tree by inserting suffixes from shortest to longest. That is, it first inserts suffix $S(n)$ into the tree, then suffix $S(n-1), \ldots$, and finally it inserts the entire string S into the tree. Weiner's algorithm constructs the suffix tree in $O(n)$ time using $O(n|\Sigma|)$ space [3, 20]. The key to Weiner's algorithm is two vectors of size Σ, a *bit vector* and a *link vector*, stored at each node. Let $L(v, x)$ specify the entry of the link vector at a node v indexed by a character x.

Definition 1. *For any character x and any node v of T, link vector $L(v, x)$ points to (internal) node u if and only if the string on the path from the root to u is $x\alpha$, where α is the string on the path from the root to v. Otherwise, $L(v, x)$ is null.*

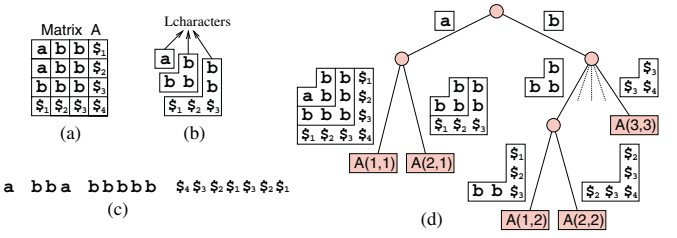


Fig. 1. (a) A matrix A, (b) the suffix $A(2,1)$ and Lcharacters composing $A(2,1)$, (c) the Lstring of A, and (d) the suffix tree of A.

For the definition of a bit vector, we refer the reader to [3, 20]. The total space of vectors is $O(n|\Sigma|)$ because a suffix tree has $O(n)$ nodes.

Weiner's algorithm can be modified to use $O(n)$ space and $O(n \log |\Sigma|)$ time for construction by eliminating a bit vector and keeping a link vector in the form of a balanced tree at each node. We omit the details.

2.2 Two-Dimensional Suffix Trees

Let A be an $n \times n$ matrix with entries defined over a finite alphabet Σ. We denote by $A[i..k, j..l]$ the submatrix of A with corners (i, j), (k, j), (i, l), and (k, l). When $i = k$ or $j = l$, we omit one of the repeated indices. For $1 \leq i, j \leq n$, the *suffix* $A(i, j)$ of A is the largest square submatrix of A that starts at position (i, j) in A. That is, $A(i, j) = A[i..i + k, j..j + k]$ where $k = n - \max(i, j)$. See Fig. 1 (a) and (b) for an example.

Let $L\Sigma = \bigcup_{i=1}^{\infty} \Sigma^{2i-1}$. We refer to the strings of $L\Sigma$ as *Lcharacters*, and we consider each of them as an atomic item. We refer to $L\Sigma$ as the *alphabet of Lcharacters*. Two Lcharacters are equal if and only if they are equal as strings over Σ. Moreover, given two Lcharacters La and Lb of equal length, we say that La is *lexicographically smaller than or equal to* Lb if and only if the string corresponding to La is lexicographically smaller than or equal to that corresponding to Lb.

We describe a linear representation of square matrices. Given $A[1..n, 1..n]$, we can divide A into n L-shaped characters. Let $a(i)$ be the concatenation of row $A[i, 1..i - 1]$ and column $A[1..i, i]$. Then $a(i)$ and $a(i)^R$ can be regarded as Lcharacters. The linearized string of matrix A, called the *Lstring* of matrix A, is the concatenation of Lcharacters $a(1)^R, \ldots, a(n)^R$. See Fig. 1 (c) for an example. Note that the linearization given here is slightly different from the original one given in [9, 12], which is $a(1)a(2) \cdots a(n)$.

The *suffix tree* of matrix A is a compacted trie over the alphabet $L\Sigma$ that represents Lstrings corresponding to all suffixes of A. See Fig. 1 (d) for an example. We refer to [9, 12] for a more formal description. Although the definition of the Lstring given here differ from that in [9, 12], our suffix trees have the same capability as suffix trees of [9, 12].

2.3 On-Line Construction

We assume that A is read *on-line* in column major order. However, our algorithm also works in case that A is read in row major order. Let $A_p = A[1..n, 1..p]$. At time $p-1$, we know A_{p-1} and nothing else about A. At time p, we get A_p by getting in input subcolumn $A[1..n, p]$.

The *GG* algorithm works in n stages, so we can limit ourselves to outline stage p, $1 \leq p \leq n$. At the end of that stage the algorithm has built a suffix tree LT_p of A_p. At the beginning of stage p, i.e., when A_{p-1} is extended into A_p, it takes in input LT_{p-1} and transforms it into LT_p. Notice that when A_{p-1} is extended into A_p, each suffix of A_{p-1} is extended by one Lcharacter. Moreover, n new suffixes of size 1×1 are created: $A[1, p], \cdots, A[n, p]$. To transform LT_{p-1} into LT_p, these changes on the matrix must be reflected on the tree. The insertion of the new suffixes in LT_{p-1} is simple. The extension of the suffixes may cause the following changes in the topology of LT_{p-1}.

(1) Leaves become internal nodes by generating new leaves.
(2) Internal nodes generate new leaves.
(3) Edges are broken by the insertion of new internal nodes and leaves.

Those changes are carried out in two phases: *Frontier Expansion* and *Internal Structure Expansion*. The first phase takes care of the changes in (1) and some other changes that do not modify the topology of the tree. The second phase takes care of changes in (2)-(3).

In the GG algorithm, each Lcharacter is regarded as an atomic unit. Comparing two Lcharacters is regarded as an atomic operation and the time complexity of the GG algorithm can be shown to be $O(n^2 \log n)\times$ (time taken by comparing two Lcharacters). Giancarlo and Guaiana [12] showed how to compare two Lcharacters in $O(\log n)$ time. Consequently, the time complexity of the GG algorithm is $O(n^2 \log^2 n)$.

We point out, omitting the details, that the auxiliary data structures allowing to compare Lcharacters in the original *GG* algorithm are *dynamic*. Indeed, they must be updated when LT_{p-1} is transformed into LT_p. Moreover, efficient support of dynamic LCA queries is an essential ingredient of that construction. In view of the results by Cole and Hariharan [4] (see next subsection), it is simple to get LCA queries in constant time over a dynamically changing tree. However that is not enough to get constant time for the comparison of Lcharacters. Indeed, as we discuss in section 3 and 4, we need a solution to the problem stated in the Introduction that takes constant time per query. That is based on a version of Weiner's algorithm [20] that works both *on-line* and *cuncurrently* for a set of strings.

2.4 Dynamic LCA Queries

Our algorithm needs LCA queries in the middle of construction. Given two nodes in a tree, LCA queries are to find the *least common ancestor (LCA)* of the two nodes. We make use of Cole and Hariharan's results [4]. They considered

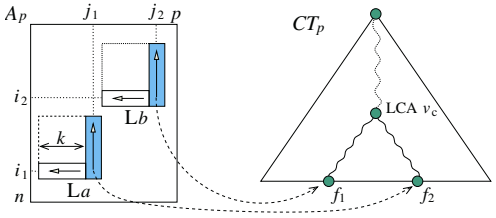


Fig. 2. Comparing column elements of two Lcharacters.

the dynamic version of the problem, i.e., maintaining a data structure which supports the following tree operations: insertion of leaves and internal nodes, deletion of internal nodes with only one child, and LCA queries. They gave an algorithm which perform the above operations in constant worst-case time.

3 Comparing Lcharacters Efficiently

We describe how to compare two Lcharacters of A_p of the same length at time p. Given two Lcharacters La and Lb, Comparing Lcharacters is to establish whether La is lexicographically smaller than or equal to Lb and to compute the length of the common longest prefix between these two strings. It can be performed in constant time using auxiliary data structures. Consequently, we can construct the suffix tree of A in $O(n^2 \log n)$ time.

We describe the auxiliary data structures. Consider the matrix A_p and let $rrow_{i,p}$ be the reversed string of the ith row for $1 \leq i \leq n$, i.e., $rrow_{i,p} = A[i,p]A[i,p-1]\ldots A[i,1]$. Let $\#_i$ be a special symbol not in the alphabet Σ such that $\#_i \prec \#_j \prec a$ for integers $i < j$ and each symbol $a \in \Sigma$. To simplify description of algorithms, we define the *extended* reversed row $err_{i,p}$ by concatenating $\#_i$ at the end of $rrow_{i,p}$. Let RT_p be the one-dimensional suffix tree representing the suffixes of $err_{1,p}, \ldots, err_{n,p}$. There is one-to-one correspondence between the leaves of RT_p and the suffixes of $err_{1,p}, \ldots, err_{n,p}$.

Similarly, let $rcol_{i,p}$ be the reversed string of the ith column for $1 \leq i \leq p$, i.e., $rcol_{i,p} = A[n,i]A[n-1,i]\ldots A[1,i]$. We define the *extended* reversed column $erc_{i,p}$ by concatenating $\#_i$ at the end of $rcol_{i,p}$. Let CT_p be the one-dimensional suffix tree representing the suffixes of $erc_{1,p}, \ldots, erc_{p,p}$. There is one-to-one correspondence between the leaves of CT_p and the suffixes of $erc_{1,p}, \ldots, erc_{p,p}$. We will show how to construct RT_p and CT_p in the next section.

Consider two Lcharacters La and Lb of length $2k+1$, from submatrices of A_p, that we need to compare. We compare separately column and row elements of Lcharacters. Without loss of generality, we assume that La is the concatenation of $A[i_1-k..i_1, j_1]^R$ and $A[i_1, j_1-k..j_1-1]^R$, and Lb is the concatenation of $A[i_2-k..i_2, j_2]^R$ and $A[i_2, j_2-k..j_2-1]^R$. We explain how to compare column elements $A[i_1-k..i_1, j_1]^R$ and $A[i_2-k..i_2, j_2]^R$. See Fig. 2. We identify a leaf f_1 of CT_p that represents the $(n-i_1+1)$st suffix of $erc_{j_1,p}$, i.e., $A[1..i_1, j_1]^R\#_{j_1}$. We find the leaf in constant time because there is one-to-one correspondence between

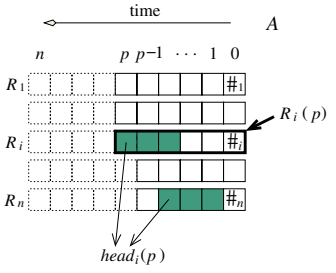


Fig. 3. n rows, $R_i(p)$, and $head_i(p)$.

the leaves of CT_p and the suffixes. We do the same for $A[i_2 - k, i_2..j_2]^R$. Let f_2 be that leaf. We find the LCA v_c of those two leaves in constant time using the results of Cole and Hariharan. Using RT_p, we perform the same operations for $A[i_1, j_1 - k..j_1 - 1]^R$ and $A[i_2, j_2 - k..j_2 - 1]^R$ to identify a node v_r analogous to v_c. Once we have found v_c and v_r, it is straightforward to establish in constant time whether La is lexicographically smaller than or equal to Lb and to compute the length of the common longest prefix between these two strings.

4 On-Line Suffix Tree

In this section, we show how to construct RT_p and CT_p. When we get subcolumn $A[1..n, p]$ at time p, we construct RT_p and CT_p from RT_{p-1} and CT_{p-1} in $O(n)$ time, respectively. For LCA queries, we maintain RT_p and CT_p in the form of Cole and Hariharan's data structure [4]. In order to transform CT_{p-1} into CT_p, we can make direct use of Weiner's suffix construction algorithm [20]. The details are given in subsection 4.2 for completeness. However, the transformation of RT_{p-1} into RT_p is not so immediate and we present it in the next subsection.

4.1 Constructing RT_p

We describe how to construct RT_p from RT_{p-1} at time p, so that LCA queries can be supported in constant time. Roughly speaking, we modify Weiner's algorithm in order to handle n strings which are input character-by-character at a time. At time p, Weiner's algorithm handles one suffix of one string while our algorithm does n suffixes, of the same length, of n strings. See Fig. 3. The remainder of this subsection is organized as follows. We first describe our modification of Weiner's algorithm and then discuss why the original algorithm would not work in overall linear time in this case.

Let $R_i[n..0]$ be $err_{i,n}$, that is, $R_i[0] = \#_i$ and $R_i[k] = A[i, k]$. We denote the suffix $R_i[k..0]$ of R_i by $R_i(k)$. We define $head_i(p)$ as the longest prefix of $R_i(p)$ that matches a substring of the n suffixes, $R_1(p), \ldots, R_{i-1}(p), R_i(p - 1), \ldots, R_n(p - 1)$. See Fig. 3. Recall that $L(v, x)$ is the entry of the link vector at a node v indexed by character x.

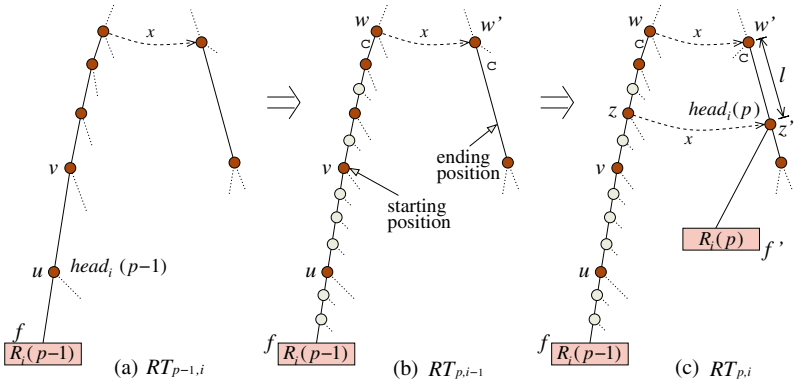


Fig. 4. Constructing $RT_{p,i}$ from $RT_{p,i-1}$ using information in $RT_{p-1,i}$.

At time p, we insert the n suffixes, $R_1(p), R_2(p), \ldots, R_n(p)$ into RT_{p-1} in order. At the beginning of step i, $R_1(p), \ldots, R_{i-1}(p)$ have already been inserted into RT_{p-1} and we inserts $R_i(p)$ at step i. We denote by $RT_{p,i}$ the suffix tree at the end of step i in time p. Note that $RT_{p,n}$ is RT_p.

We describe how to insert the suffix $R_i(p)$ into $RT_{p,i-1}$ to produce $RT_{p,i}$ at step i in time p. For this, we first find the end of the path labeled with $head_i(p)$ in tree $RT_{p,i-1}$. We use the information which is created when $R_i(p-1)$ was inserted at step i in time $p-1$. Let x be the first character of the suffix $R_i(p)$, i.e., $x = R_i[p]$. Let f and u be the leaf and internal node such that the strings on the paths from the root to f and u are $R_i(p-1)$ and $head_i(p-1)$, respectively. Let v be the parent node of u in $RT_{p-1,i}$ if u is not the root node. See Fig. 4. Note that v may not the parent node of u in $RT_{p,i-1}$ because $RT_{p,i-1}$ is the tree which is created by inserting the n suffixes, $R_{i+1}(p-1), \ldots, R_n(p-1), R_1(p), \ldots, R_{i-1}(p)$, into $RT_{p-1,i}$.

The Algorithm at Step i in Time p

1. (a) If $L(u, x)$ is not null, say z',
   then $head_i(p)$ ends at z'. Go to Stage 3.
   (b) If $L(u, x)$ is null and u is the root,
   then set w and w' to be the root, and α to be the string $R_i(p)$.
   (c) If $L(u, x)$ is null and u is not the root,
   then start at node v of $RT_{p,i-1}$ and walk toward the root searching for the first node w on walk such that $L(w, x)$ is not null.
      i. If w was found such that $L(w, x)$ is not null,
      then set w' to be $L(w, x)$ and α to be the string on the path from w to f.
      ii. Otherwise, set w and w' to be the root and α to be the string $R_i(p)$.
2. Search for the edge e out of w' whose first character is c, where c is the first character of α.

(a) If such edge e exists,
then find the length l of the longest common prefix between α and β, where β be the string on edge e. Then, $head_i(p)$ ends exactly at l characters below w' on edge e.
(b) Otherwise, $head_i(p)$ ends at node w'.

3. If a node already exists at the end of $head_i(p)$, then let z' denote that node; otherwise, create a node z' at the end of $head_i(p)$. Create a new leaf f' representing $R_i(p)$ and a new edge connecting z' and f.

* How to update link vectors.
Updates for link vectors occur only in case that v' is newly created. Let z be the node on the path from w to f such that the string on the path from w to z is $\alpha[1..l]$ if w' is not the root, $\alpha[2..l]$ otherwise. It is guaranteed that node z exists in $RP_{p,i-1}$. We update $L(z, x)$ to point to z', which is the only update needed.

The key of our algorithm is as follows. Differently with Weiner's algorithm to construct the suffix tree for one string, many nodes are created newly between inserting $R_i(p-1)$ and inserting $R_i(p)$ due to inserting suffixes of other rows. We call such nodes *intra-nodes*. In Fig. 4, intra-nodes are represented by white nodes. Because all entries of link vectors at intra-nodes are null, we do not need to check link vectors of intra-nodes. Besides, traversing intra-nodes between v and f increases time complexity. Thus, we skip intra-nodes between v and f by first checking $L(u, x)$, and then checking $L(v, x)$ if $L(u, x)$ is null and u is not the root (i.e., Stage 1 (c)). The other intra-nodes are treated as ordinary nodes. The correctness of our algorithm can be proved as that of Weiner's algorithm. We omit the details.

Lemma 1. *We can insert all suffixes of a row R_i in $O(n)$ time. That is, at time p, we can insert the suffix $R_i(p)$ into the suffix tree $RT_{p,i-1}$ in amortized constant time.*

Corollary 1. *At time p, we get the suffix tree RT_p from RT_{p-1} in amortized $O(n)$ time. The total time to create RT_n is $O(n^2)$.*

4.2 Constructing CT_p

Constructing CT_p is easier than constructing RT_p. Let C_p be the concatenation of strings $erc_{p,p}, erc_{p-1,p}, \ldots, erc_{1,p}$. CT_p is the suffix tree of C_p. We have the suffix tree of C_{p-1} at the beginning of time p. When we get subcolumn $A[1..n, p]$, we add $erc_{p,p}$ in front of C_{p-1}. In order to get the suffix tree CT_p of C_p, we insert the first $(n+1)$ suffixes of $erc_{p,p}C_{p-1}$ into CT_{p-1} by Weiner's algorithm. It take $O(n)$ time.

Lemma 2. *At time p, we get the suffix tree CT_p from CT_{p-1} in $O(n)$ time. The total time to create CT_n is $O(n^2)$.*

Theorem 1. *The total time to construct on-line the two-dimensional suffix tree for an $n \times n$ matrix A is $O(n^2 \log n)$.*

References

1. A. Amir and M. Farach-Colton. Two-dimensional matching algorithms. In A. Apostolico and Z. Galil, editors, *Pattern Matching Algorithms*, pages 267–292. Oxford University Press, Oxford, 1997.
2. A. Apostolico. The myriad virtues of subword trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, pages 85–95. Springer, 1985.
3. M.T. Chen and J. Seiferas. Efficient and elegant subword tree construction. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12 of *F*, pages 97–107. NATO Advanced Science Institutes, Spring-Verlag, 1985.
4. R. Cole and R. Hariharan. Dynamic LCA queries on trees. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 235–244, 1999.
5. R. Cole and R. Hariharan. Faster suffix tree construction with missing suffix links. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 407–415, 2000.
6. M. Crochemore and W. Rytter. *Jewels of Stringology*. World Scientific Publishing, Singapore, 2002.
7. M. Farach-Colton, P. Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *Journal of the ACM*, 47(6):987–1011, 2000.
8. R. Giancarlo. An index data structure for matrices, with applications to fast two-dimensional pattern matching. In *Proceedings of Workshop on Algorithm and Data Structures*, pages 337–348. Springer-Verlag, 1993.
9. R. Giancarlo. A generalization of the suffix tree to square matrices, with application. *SIAM Journal on Computing*, 24(3):520–562, 1995.
10. R. Giancarlo and R. Grossi. On the construction of classes of suffix trees for square matrices: Algorithms and applications. *Information and Computation*, 130(2):151–182, 1996.
11. R. Giancarlo and R. Grossi. Suffix tree data structures for matrices. In A. Apostolico and Z. Galil, editors, *Pattern Matching Algorithms*, chapter 11, pages 293–340. Oxford University Press, Oxford, 1997.
12. R. Giancarlo and D. Guaiana. On-line construction of two-dimensional suffix trees. *Journal of Complexity*, 15(1):72–127, 1999.
13. D. Gusfield. *Algorithms on Strings, Tree, and Sequences*. Cambridge University Press, Cambridge, 1997.
14. D. K. Kim, Y. A. Kim, and K. Park. Generalizations of suffix arrays to multi-dimensional matrices. *Theoretical Computer Science*, 302(1-3):401–416, 2003.
15. D. K. Kim and K. Park. Linear-time construction of two-dimensional suffix trees. In *Proceedings of the 26th International Colloquium on Automata, Languages, and Programming*, pages 463–372, 1999.
16. U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
17. E. M. McCreight. A space-economical suffix tree construction algorithms. *Journal of the ACM*, 23(2):262–272, 1976.
18. J. A. Storer. Lossless image compression using generalized LZ1-type methods. In *Proceedings of Data Compression Conference*, pages 290–299, 1996.
19. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
20. P. Weiner. Linear pattern matching algorithms. In *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

Min-Energy Voltage Allocation for Tree-Structured Tasks

Minming Li[1,⋆], Becky Jie Liu[2], and Frances F. Yao[2]

[1] Department of Computer Science, Tsinghua University
liminming98@mails.tsinghua.edu.cn
[2] Department of Computer Science, City University of Hong Kong
jliu@cs.cityu.edu.hk, csfyao@cityu.edu.hk

Abstract. We study job scheduling on processors capable of running at variable voltage/speed to minimize energy consumption. Each job in a problem instance is specified by its arrival time and deadline, together with required number of CPU cycles. It is known that the minimum energy schedule for n jobs can be computed in $O(n^3)$ time, assuming a convex energy function. We investigate more efficient algorithms for computing the optimal schedule when the job sets have certain special structures. When the time intervals are structured as trees, the minimum energy schedule is shown to have a succinct characterization and is computable in time $O(P)$ where P is the tree's total path length. We also study an on-line average-rate heuristics AVR and prove that its energy consumption achieves a small constant competitive ratio for nested job sets and for job sets with limited overlap. Some simulation results are also given.

1 Introduction

Portable electronic devices have in recent years seen a dramatic rise in availability and widespread use. This is partly brought on by new technologies enabling integration of multiple functions on a single chip (SOC). However, with increasing functionality also comes ever greater demand for battery power, and energy efficient implementations have become an important consideration for portable devices.

Generally speaking, the main approach is to trade execution speed for lower energy consumption while still meeting all deadlines. A number of techniques have been applied in embedded systems to reduce energy consumption. Modes such as idle, standby and sleep are already available in many processors. More energy savings can be achieved by applying Dynamic Voltage Scaling (DVS) techniques on variable voltage processors, such as the Intel *SpeedStep* technology [2] currently used in Intel's notebooks. With the newest *Foxon* technology

⋆ This work is supported by National Natural Science Foundation of China (60135010), National Natural Science Foundation of China (60321002) and the Chinese National Key Foundation Research & Development Plan (2004CB318108).

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 283–296, 2005.

(announced in February 2005), future Intel server chips can choose from as many as 64 speed grades, up from two or three in SpeedStep.

The associated scheduling problem for variable voltage processors has generated much interest, and an extensive literature now exists on this research topic. One of the earliest models for energy-efficient scheduling was introduced by Yao, Demers and Shenker in [1]. They described a minimum-energy off-line preemptive scheduling algorithm, with no restriction on the power consumption function except convexity. Also, two on-line heuristics AVR (Average Rate) and OPA (Optimal Available) were introduced, and it was shown that AVR has a competitive ratio of at most 8 for all job sets.

Under various related models and assumptions, more theoretical research has been done on minimum energy scheduling. For jobs with fixed priority, it was shown to be NP-hard to calculate the min-energy schedule and an FPTAS was given for the problem by Yun and Kim [6]. For discrete variable voltage processors, a polynomial time algorithm for finding the optimal schedule was given by Kwon and Kim [3]. Recently a tight competitive ratio of 4 was proven for the Optimal Available heuristic (OPA) [7]. Another related model which focuses on power down energy was considered in [9].

On the practical side, the problem has been considered under different systems constraints. For example, in [5] a task slowdown algorithm that minimizes energy consumption was proposed, taking into account resource standby energy as well as processor leakage. By considering limitations of current processors, such as transition overhead or discrete voltage levels, it was shown how to obtain a feasible (although non-optimal) schedule [4].

In this paper, we present efficient algorithms for computing the optimal schedules when the time intervals of the tasks have certain natural structures. These include tree-structured job sets which can arise from executing recursive procedure calls, and job sets with limited time overlap among tasks. We derive succinct characterizations of the minimum-energy schedules in these cases, leading to efficient algorithms for computing them. For general trees we obtain an $O(P)$ algorithm where P is the tree's total path length. In special cases when the tree is a nested chain or has bounded depth, the complexity reduces to $O(n)$. We also study the competitive ratio of on-line heuristic AVR for common-deadline job sets and limited-overlap job sets. A tight bound of 4 is proved in the former case, and an upper bound of 2.72 is proved in the latter case. Finally, we establish a lower bound of $\frac{17}{13}$ on the competitive ratio for any online schedule, assuming that time is discrete.

The significance of our work is twofold. First, the cases we consider, such as tree-structured tasks or common-deadline tasks, represent common job types. A thorough understanding of their min-energy schedules and effective heuristics can serve as useful tools for solving other voltage scheduling problems. Secondly, the characterization of these optimal solutions give rise to nice combinatorial problems of independent interest. For example, the optimal voltage scheduling for tree job sets can be viewed as a special kind of weight-balancing problem on trees (see Section 3).

The paper is organized as follows. We first review the scheduling model, off-line optimal schedule, and on-line AVR heuristic in Sections 2. In Section 3, we consider tree job sets and develop effective characterizations and algorithms for finding the optimal schedule. We also point out two special cases, the nested chain and the common deadline cases and give particularly compact algorithms for them. Analysis of competitive ratio is presented in Section 4 and lower bound of competitive ratio is discussed in Section 5. After presenting some simulation results in Section 6, we finish with concluding remarks and open problems in Section 7.

2 Preliminaries

We first review the minimum-energy scheduling model described in [1], as well as the off-line optimal scheduling algorithm and AVR online heuristic. For consistency, we adopt the same notions as used in [1].

2.1 Scheduling Model

Let J be a set of jobs to be executed during time interval $[t_0, t_1]$. Each job $j_k \in J$ is characterized by three parameters.

- a_k arrival time,
- b_k deadline, $(b_k > a_k)$
- R_k required number of CPU cycles.

A schedule S is a pair of functions $\{s(t), job(t)\}$ defined over $[t_0, t_1]$. Both $s(t)$ and $job(t)$ are piecewise constant with finitely many discontinuities.

- $s(t) \geq 0$ is the processor speed at time t,
- $job(t)$ defines the job being executed at time t (or idle if $s(t) = 0$).

A feasible schedule for an instance J is a schedule S that satisfies $\int_{a_k}^{b_k} s(t)\delta(job(t), j_k)dt = R_k$ for all $j_k \in J$ (where $\delta(x, y)$ is 1 if $x = y$ and 0 otherwise). In other words, S must give each job j the required number of cycles between its arrival time and deadline(with perhaps intermittent execution). We assume that the power P, or energy consumed per unit time, is a convex function of the processor speed. The total energy consumed by a schedule S is $E(S) = \int_a^b P(s(t))dt$.

The goal of the scheduling problem is to find, for any given problem instance, a feasible schedule that minimizes $E(S)$. We remark that it is sufficient to focus on the computation of the optimal speed function $s(t)$; the related function $job(t)$ can be obtained with the earliest-deadline-first (EDF) principle.

2.2 The Minimum Energy Scheduler

We consider the off-line version of the scheduling problem and give a characterization of an energy-optimal schedule for any set of n jobs.

The characterization will be based on the notion of a *critical interval* for J, which is an interval in which a group of jobs must be scheduled at maximum constant speed in any optimal schedule for J. The algorithm proceeds by identifying such a critical interval for J, scheduling those 'critical' jobs, then constructing a subproblem for the remaining jobs and solving it recursively. The optimal $s(t)$ is in fact unique, whereas $job(t)$ is not always so. The details are given below.

***Definition* 1** *Define the intensity of an interval $I = [z, z']$ to be $g(I) = \frac{\sum R_k}{z'-z}$ where the sum is taken over $J_I = \{$all jobs j_k with $[a_k, b_k] \subseteq [z, z']\}$.*

Clearly, $g(I)$ is a lower bound on the average processing speed, $\int_z^{z'} s(t)dt/(z'-z)$, that is required by any feasible schedule over the interval $[z, z']$. By convexity of the power function, any schedule using constant speed $g(I)$ over I is necessarily optimal on that interval. A critical interval I^* is an interval with maximum intensity $\max g(I)$ among all intervals I. It can be shown that the jobs in J_{I^*} allow a feasible schedule at speed $g(I^*)$ with the EDF principle. Based on this, Algorithm OS for finding the optimal schedule is given below and it can be implemented in $O(n^3)$ time [1].

Algorithm 1 *OS (Optimal Schedule)*

Input: a job set J
Output: Optimal Voltage Schedule S

```
repeat
  Select I* = [z, z'] with s = max g(I)
  Schedule the jobs in J_{I*} at speed s by EDF policy
  J ← J − J_{I*}
  for all j_k ∈ J do
    if b_k ∈ [z, z'] then
      b_k ← z
    else if b_k ≥ z' then
      b_k ← b_k − (z' − z)
    end if
    Reset arrival times similarly
  end for
until J is empty
```

2.3 On-Line Scheduling Heuristics

Associated with each job j_k are its *average-rate* $d_k = \frac{R_k}{b_k - a_k}$ and the corresponding step function

$$d_k(t) = \begin{cases} d_k \text{ if } t \in [a_k, b_k] \\ 0 \quad \text{elsewhere.} \end{cases}$$

Average Rate Heuristic computes the processor speed as the sum of all available jobs' average-rate. See Figure 1. At any time t, processor speed is set to be

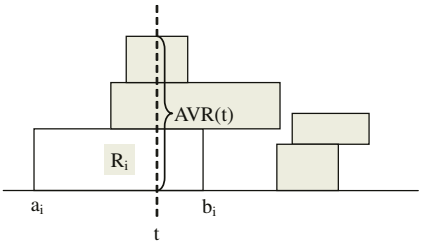


Fig. 1. Example of AVR heuristic

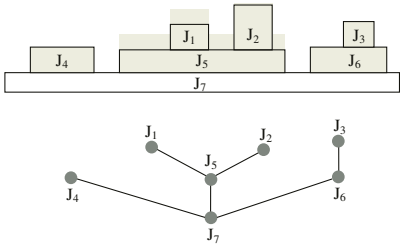


Fig. 2. Example of a tree job set

$s(t) = \sum_k d_k(t)$. Then, it uses the EDF policy to choose among available jobs for execution. Obviously this approach yields a feasible schedule.

Since the analysis of competitive ratio depends on the precise form of $P(s)$, our analysis is conducted under the assumption that $P(s) = s^2$. For a given job set J, let $OPT(J)$ denote the energy consumption of the optimal schedule, and let $AVR(J) = \int (\sum_k d_k(t))^2 dt$ denote the energy consumption of the AVR heuristic schedule. The competitive ratio of the heuristic is defined as the least upper bound of $AVR(J)/OPT(J)$ over all J.

It has been proved in [1] that AVR heuristic has a competitive ratio of at most 8 for any job set.

3 Optimal Voltage Schedule for Tree Job Sets

We consider the scheduling instance when the job intervals are properly nested as in a tree structure. The motivations are twofold: 1) such job sets arise naturally in practice, e.g. in the execution of recursively structured programs; 2) the characterization of the optimal speed function is nontrivial and leads to interesting combinatorial problems of independent interest.

3.1 Characterization of Optimal Schedule for Trees

***Definition* 2** *A job set J is called a tree job set if for any pair of job intervals I_j and I_k, one of the following relations holds: $I_j \cap I_k = \emptyset$, $I_j \subseteq I_k$ or $I_k \subseteq I_j$.*

For a tree job set, the inclusion relationship among job intervals can be represented graph-theoretically by a tree where each node corresponds to a job. See Figure 2. Job j_i is a descendant of job j_k iff $I_i \subseteq I_k$; if furthermore no other job $j_{i'}$ satisfies $I_i \subseteq I_{i'} \subseteq I_k$ then job j_i is a child of job j_k.

Interesting special cases of trees include jobs forming a single path (a nested chain), and jobs sharing a common deadline (or symmetrically, common arrival time). We will present linear algorithms for these cases in the next subsection, after first describing properties and algorithms for the general tree case. As remarked before, it is sufficient to focus on the computation of the optimal

speed function $s(t)$; the related function $job(t)$ can be obtained with the EDF principle.

We will prove a key lemma that is central to constructing optimal schedules for tree job sets. Suppose an optimal schedule has been given for a tree job set without its root node, we consider how to update the existing optimal schedule when a root node is added.

Consider any job set J consisting of n jobs (not necessary tree-structured), and an additional new job j_{n+1} with the property that $[a_{n+1}, b_{n+1}] \supseteq [a_k, b_k]$ for any $j_k \in J$. We will show that the optimal schedule $s'(t)$ for job set $J' = J \cup \{j_{n+1}\}$ is uniquely determined from the optimal schedule $s(t)$ of J and description of j_{n+1}. That is, information such as $[a_k, b_k]$ and R_k for $j_k \in J$ is not needed for computing $s'(t)$ from $s(t)$. This property will enable us to construct the optimal schedule for a tree job set efficiently in a bottom-up procedure.

To prove the above claim, we compare the selection of critical intervals for $s'(t)$ versus that for $s(t)$. Suppose $s(t)$ consists of m critical intervals $I_1^*, \ldots, I_m^*$ with lengths $l_1, \ldots, l_m$ and speeds $s_1 > \cdots > s_m$. Comparing the computation of $s'(t)$ by Algorithm OS versus that of $s(t)$, we note that the only new candidate for critical intervals in each round is $I_{n+1} = [a_{n+1}, b_{n+1}]$. (Due to the fact $[a_{n+1}, b_{n+1}] \supseteq [a_k, b_k]$, no other intervals of the form $[a_{n+1}, b_k]$ or $[a_k, b_{n+1}]$ is a feasible candidate.) Moreover, as soon as I_{n+1} is selected as the critical interval, all currently remaining jobs will be executed (since their intervals are contained in I_{n+1}) and Algorithm OS will terminate.

By examining the intensity $g(I_{n+1})$ of interval I_{n+1} in each round i and comparing it with the speed s_i, we can determine exactly in which round I_{n+1} will be selected as the critical interval. This index i will be referred to as the *terminal index* (of job j_{n+1} relative to $s(t)$). To find the terminal index i, it is convenient to start with the maximum $i = m + 1$ and search backwards. Let $g_i(I_{n+1})$ denote the intensity of I_{n+1} as would be calculated in the i-th round of Algorithm OS. Using $w_k = s_k l_k$ to denote the workload executed in the k^{th} critical interval of $s(t)$ for $k = 1, \ldots, m$, and letting $w_{m+1} = R_{n+1}$ and $l_{m+1} = |I_{n+1}| - \sum_{k=1}^{m} l_k$, we can write $g_i(I_{n+1})$ as $g_i(I_{n+1}) = (\sum_{k=i}^{m+1} w_k)/(\sum_{k=i}^{m+1} l_k)$.

It follows that the terminal index is the largest i, $1 \le i \le m+1$, for which the following holds: $g_i(I_{n+1}) \ge s_i$ and $g_{i-1}(I_{n+1}) < s_{i-1}$ (where we set $s_0 = \infty$, $g_0(I_{n+1}) = 0$, and $s_{m+1} = 0$ as boundary conditions).

We have proven the following main lemma for tree job sets.

Lemma 1 *Let the optimal schedule $s(t)$ for a job set J be given, consisting of speeds $s_1 > \cdots > s_m$ over intervals $I_1^*, \ldots, I_m^*$. If a new job j_{n+1} satisfies $[a_{n+1}, b_{n+1}] \supseteq [a_k, b_k]$ for all $j_k \in J$, then the optimal schedule $s'(t)$ for $J \cup \{j_{n+1}\}$ consists of speeds $s'_1 > s'_2 > \cdots > s'_i$ where*
1) i is the terminal index of j_{n+1} relative to $s(t)$,
2) critical intervals from 1 up to $i-1$ are identical in $s(t)$ and $s'(t)$, and
3) the i-th critical interval for $s'(t)$ has speed $s'_i = g_i(I_{n+1})$ over $I_{n+1} - \bigcup_{k=1}^{i-1} I_k^*$.

A procedure corresponding to the above lemma is given in Algorithm *Merge*, which updates an optimal schedule when a root node is added. The exact method

for finding the terminal index will be discussed in the next subsection. Based on *Merge*, we obtain a recursive algorithm for finding the optimal schedule for a tree job set as given in Algorithm *OST*. We also observe the following global property of a tree-induced schedule.

Lemma 2 *In the optimal schedule for a tree job set, the execution speeds of jobs along any root-leaf path form a non-decreasing sequence.*

Proof. By Lemma 1, for a tree job set J' consisting of a node j_{n+1} and all of its descendants, the execution speed of j_{n+1} is the minimum among J' since it defines the last critical interval. The lemma holds by treating every node as the root of some subtree.

3.2 Finding Terminal Indices for Trees

Algorithm OST gives a recursive procedure for constructing optimal schedule for a tree job set. The important step is to carry out $Merge(S_{J-\{r\}}, r)$ at every node r of the tree by finding the correct terminal index i. Naively, it would seem that sorting the execution speeds of all of $r's$ descendants is necessary for finding the terminal index quickly. It turns out that sorting (an $O(n \log n)$ process) can be replaced by median-finding (an $O(n)$ process [8]) as we show next. This enables us to achieve $O(P)$ complexity for the overall algorithm where P is the tree's total path length. For trees of bounded depth this gives an $O(n)$ algorithm. In the following discussion, we denote the optimal schedule of job set J as S_J.

Algorithm 2 *Merge*

Input: Optimal schedule S for J, new job j_{n+1}
Output: Optimal schedule S' for $J' = J \cup \{j_{n+1}\}$

$S' \leftarrow S$
$i \leftarrow Find(S, j_{n+1})$ {find terminal index i}
In S', replace $s_i, s_{i+1}, \ldots, s_m$ with $s'_i = g_i(I_{n+1})$
Return S'

Algorithm 3 *OST (Optimal Scheduling for Tree)*

Input: root r of tree job set J
Output: Optimal Schedule S_J for J

initialize $S_{J-\{r\}}$ to be $\emptyset$
for all Children ch_k of r **do**
  $S_{J-\{r\}} \leftarrow S_{J-\{r\}} \cup OST(ch_k)$
end for
$S_J \leftarrow Merge(S_{J-\{r\}}, r)$
Return S

Theorem 1 *Algorithm OST can compute an optimal schedule for any tree job set in $O(P)$ time where P is the total path length of the tree.*

Proof. For every $Merge$ operation, we can find the terminal index by performing a binary search on the median speed in S_{J-r}. That is, we find the median speed s_k and then decide in which half to search further. See Algorithm 4. Finding the median of a list of t items costs $O(t)$ time. Calculation of the associated $g(I_{n+1})$ value is also $O(t)$. The total cost of a binary search for the terminal index thus amounts to a geometric series whose sum is bounded by $O(t)$. Therefore, the cost of $Merge(S_{J-\{r\}}, r)$ is proportional to the number of descendant nodes of r. Hence the total cost of $Merge$ over all nodes of the tree is upper bounded by the tree's total path length.

Algorithm 4 *Find (by Median Search)*

Input: Schedule S consisting of speed $\{s_1, s_2, \cdots, s_m\}$ in unsorted manner, new job j_{n+1}

Output: Index i such that $g_i(I_{n+1}) \geq s_i$ and $g_{i-1}(I_{n+1}) < s_{i-1}$.

```
s_0 ← ∞
s_{m+1} ← 0
Find median value s_k in S
while k isn't the terminal index do
  if g_k(I_{n+1}) < s_k then
    S ← {s_j | j > k, s_j ∈ S}
  end if
  if g_{k-1}(I_{n+1}) ≥ s_{k-1} then
    S ← {s_j | j < k, s_j ∈ S}
  end if
  Find median speed s_k in S
end while
Return k
```

3.3 Finding Terminal Indices for Chains

For trees of depth $O(n)$, the above algorithm can have worst case complexity $O(n^2)$. However, we will show that for a nested chain of n jobs (corresponding to a single path of depth n), its optimal schedule can still be computed in $O(n)$ time. Here, instead of using repeated median-finding, Algorithm *Merge* will keep the speeds sorted and use a linear search to find the terminal index. We note that, without loss of generality, the n nested jobs can be first shifted so they all have a common deadline. (This is because the intersection relationship among time intervals have not been altered.) See Figure 3. Thus it is sufficient to describe an $O(n)$ algorithm for job sets with a common deadline.

Theorem 2 *The optimal schedule for a job set corresponding to a nested chain can be computed in $O(n)$ time.*

Proof. Let the job intervals be $[a_{n+1}, b] \supseteq [a_n, b] \supseteq \cdots \supseteq [a_1, b]$. We implement $Find(S_{J-\{j_{n+1}\}}, j_{n+1})$ with a linear search for the terminal index, starting with

the lowest speed in $S_{J-\{j_{n+1}\}}$. The key observation is that, as can be proven by induction, the speed function $s(t)$ is piecewise increasing from left to right. Hence the search for the terminal index can proceed from left to right, computing $g_k(I_{n+1})$ one by one from the smallest s_k. Notice that computing $g_k(I_{n+1})$ with knowledge of $g_{k-1}(I_{n+1})$ only costs constant operations. Furthermore, if we needed to compute g for u consecutive $k's$ before arriving at the right terminal index, then the total number of critical intervals will also have decreased by $u-1$. Suppose the Merge operation for the j-th job in the chain (starting from the leaf) computes g for u_j times, we have $\sum_{k=1}^{n}(u_k-1) \leq n$, which implies that $\sum_{k=1}^{n} u_k \leq 2n$. Therefore, the algorithm can finish computing the optimal schedule in $O(n)$ time.

The linear search procedure described above is given in Algorithm 5, and one execution of this *Find* procedure is illustrated in Figure 4.

Algorithm 5 *Find (by Linear Search)*

Input: Schedule S = $\{s_1 > s_2 > \cdots > s_m\}$, new job j_{n+1}
Output: Index i such that $g_i(I_{n+1}) \geq s_i$ and $g_{i-1}(I_{n+1}) < s_{i-1}$.

```
s_0 ← ∞
s_{m+1} ← 0
for k = m downto 1 do
  if g_k(I_{n+1}) < s_k then
    Return k + 1
  end if
end for
```

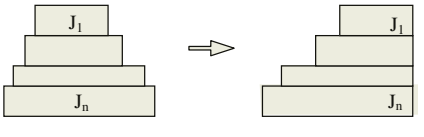


Fig. 3. Transforming chain job set into common deadline job set

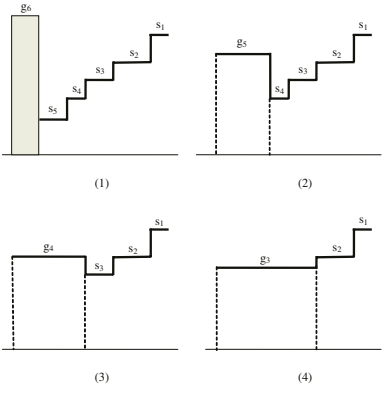


Fig. 4. Scheduling of a set of jobs with common deadline

3.4 A Weight Balancing Problem for Trees

We have presented two different strategies for implementing the *Find* operation on trees, as a subroutine used for computing the optimal voltage schedule. We

can formulate the problem as a pure weight balancing problem for trees, whose solution will provide alternative algorithms for computing the optimal schedule.

We start with a tree where each node is associated with a pair of weights (w_k, l_k). The goal is to adjust the weights so that the ratio $s_k = w_k/l_k$ along any root-leaf path will be monotonically non-decreasing (see Lemma 2). The rule for modifying the weights is to proceed recursively and, at each node r, 'merge' $r's$ weights with those of its descendants with smallest s_k so that their new 'average' ratio, as defined by $(\sum w_k)/\sum l_k)$, will satisfy the monotonicity condition. The challenge is to find a suitable data structure that supports the selection of $r's$ descendants for the weight balancing . Two different solutions to this problem were considered in Theorem 1 and 2 respectively. Are there other efficient methods?

4 Analysis of Competitive Ratio

We will analyze the performance of AVR versus OPT for several types of job sets. For convenience of reference, we state the definition of these job sets in the following.

***Definition* 3** *A Job set J is called*

i) a chain job set if $a_1 \leq a_2 \leq ... \leq a_n$ and $b_1 \geq b_2 \geq ... \geq b_n$

ii) a common deadline job set if $a_1 \leq a_2 \leq ... \leq a_n$ and $b_1 = b_2 = ... = b_n$

iii) a two-overlap job set if $I_i \cap I_{i+2} = \emptyset$ and $a_1 \leq a_2 \leq ... \leq a_n$ and $b_1 \leq b_2 \leq ... \leq b_n$.

4.1 Chain Job Set

Theorem 3 *For any nested chain job set J, $AVR(J) \leq 4\ OPT(J)$.*

This bound of 4 is tight, as an example J provided in [1] actually achieves $AVR(J) = 4\ OPT(J)$. In this example, the ith job has interval $[0, 1/n]$ and density $d_i = (n/i)^{3/2}$, this job set has competitive ratio 4 when $n \to \infty$.

It is obvious that transforming a chain job set into a common deadline job set by shifting preserves both $AVR(J)$ and $OPT(J)$, hence does not affect the competitive ratio. See Fig. 3. Thus we only need to focus on the competitive ratio for the common deadline case. Given a common deadline job set J, the algorithm in Theorem 2 will produce an optimal schedule with exactly one execution interval for each job $j_i \in J$. Denote the execution interval by $[c_i, c_{i+1}]$ where $c_1 = a_1$, $c_i \geq a_i$ and $c_{n+1} = b$. Given J, define J' to be the same as J except $a'_i = c_i$ for all i. We call J' the *normalized* job set for J.

We make use of the following algebraic relation; the proof is omitted here.

Lemma 3 *Let X and X' be two positive constants such that $\int_a^b X = \int_{a'}^b X'$ where $a \leq a' \leq b$. If $Y(t)$ is a monotone function such that $Y(t) \leq Y(t')$ for $t \leq t'$, then $\int_a^b (X + Y(t))^2 \leq \int_{a'}^b (X' + Y(t))^2 + \int_a^{a'} Y(t)^2$.*

Lemma 4 *Given a common deadline job set J, let J' be the normalized job set for J. Then we have*
1) $OPT(J) = OPT(J')$
2) $AVR(J) \leq AVR(J')$.

Proof. Property 1) is straightforward by the definition of J'. Property 2) can be proved by applying lemma 3 to the jobs inductively.

Proof of Theorem 3. We first convert J into a common deadline job set. Let J' be the normalized job set for J. By lemma 4, $AVR(J) \leq AVR(J')$ and $OPT(J) = OPT(J')$. According to Theorem 2 in [1], this will result in a competitive ratio of at most 4 for J'. Combining with $AVR(J) \leq AVR(J')$, we obtain $AVR(J) \leq 4\ OPT(J)$.

4.2 Two-Overlap Job Set

We first consider the simple case of a two-job instance and show that $AVR(J) \leq 1.36\ OPT(J)$ (proof omitted here). It is then used as the basis for the n-job case.

Lemma 5 *For any job set consisting of two jobs, $AVR(J) \leq 1.36\ OPT(J)$.*

Theorem 4 *For any two-overlap job set, $AVR(J) \leq 2.72\ OPT(J)$.*

Proof. Denote the two-job instance $\{j_i, j_{i+1}\}$ as J_i for $0 \leq i \leq n$. (We also introduce two empty jobs j_0 and j_{n+1}.) We have $AVR(J) = \sum_{i=0}^{n} AVR(J_i) - \sum_{i=1}^{n} d_i^2 t_i$. On the other hand, using the result for two-job sets, we have $\sum_{i=0}^{n} AVR(J_i) \leq 1.36 \sum_{i=0}^{n} OPT(J_i)$. We observe that any two consecutive jobs in a two-overlap job set must use more energy in the optimal schedule than when they are scheduled alone. Thus, $\sum_{i=0}^{n} OPT(J_i) \leq 2\ OPT(J)$. Combining the above three relations, we obtain $AVR(J) \leq 2.72\ OPT(J) - \sum_{i=1}^{n} d_i^2 t_i$ which proves the theorem.

5 Lower Bound for Online Schedules

To prove a lower bound on the competitive ratio of all online schedules, we make the assumption that the processor time comes in discrete units, i.e. the processor must maintain the same speed over each time unit. Given any online scheduler, we will construct a two-job instance for which the scheduler's performance is no better than $\frac{17}{13}$ times optimal.

The first job arrives at time 0 and its interval lasts for two time units. Its requirement is two CPU cycles. Suppose the online schedule allocates two CPU cycles to the first job on the first time unit. We then just let the second job be an empty job, and the schedule's cost is already 2 times optimal.

Suppose the online schedule allocates one CPU cycle to the first job on the first time unit. We construct the second job as follows: it starts from the second time unit and lasts for one time unit and requires 3 CPU cycles. In this case the online schedule's cost is $\frac{17}{13}$ times the optimal. This proves that $\frac{17}{13}$ is a lower bound on the competitive ratio of all online heuristics.

6 Simulation Results

We have simulated the performance of AVR online heuristic in three different types of job sets: general, two-overlap and common deadline job sets. The following data are collected from 1000 randomly generated job sets of each type. Each job set consists of 100 random jobs: the arrival times and deadlines are uniformly distributed over a continuous time interval $[0, 100]$ in the general case, and suitably constrained in the other two cases. The required CPU cycles of each job is chosen randomly between 0 and 200.

Average, maximum and minimum competitive ratios for each of the three cases are shown in Table 5. For the general case, the maximum competitive ratio we observed is 1.47, which is far better than the theoretical bound of 8. The minimum observed ratio is always close to 1. Best among the three, the two-overlap case is where AVR excels, achieving average ratio of 1.16 and maximum ratio of 1.21. For the common deadline case, the maximum ratio encountered of 2.25 is also much better than the bound of 4 proved in Theorem 3.

Type of Job Set	Average	Maximum	Minimum
General	1.215	1.469	1.007
Two-overlap	1.160	1.206	1.088
Common Deadline	1.894	1.255	1.113

Fig. 5. Summary of competitive ratios from simulations

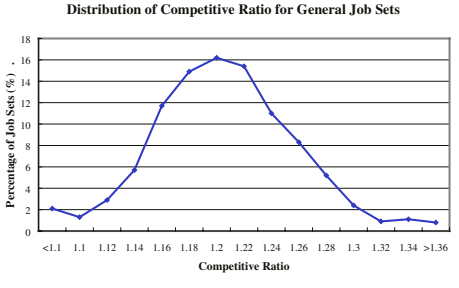


Fig. 6. Simulation result of competitive ratio for general job sets

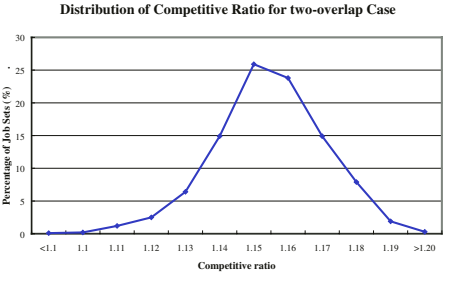


Fig. 7. Simulation result of competitive ratio for two-overlap job sets

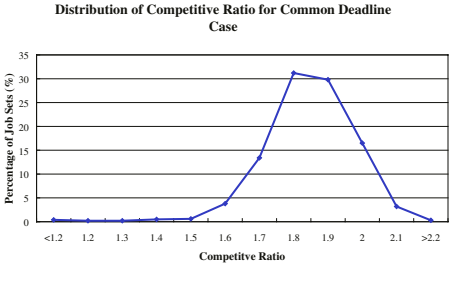


Fig. 8. Simulation result of competitive ratio for common-deadline job sets

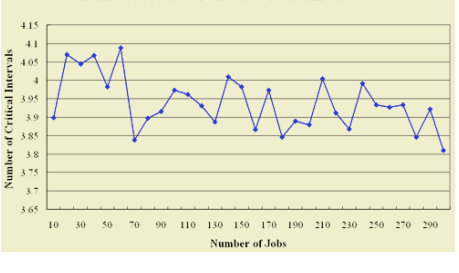


Fig. 9. Number of jobs vs. number of critical intervals

The detailed distributions of the competitive ratio obtained from the simulations are give in Figure 6, 7 and 8. The data suggest that the distributions are close to normal for all three types of job sets, with standard deviations of 0.0528, 0.0162 and 0.1336 respectively.

In the second simulation, we look for the growth of the number of critical intervals with respect to the number of jobs in the general case. For each n between 10 and 300, we randomly generate a set of n jobs, and then compute the average number of critical intervals over 1000 such job sets for each n. Quite surprisingly, this average number does not seem to grow noticeably with the number of jobs. According to Figure 9, the average number of critical intervals always lies within the range from 3.8 to 4.1 for any n between 10 to 300, with lowest value 3.81 for $n = 300$ and highest value 4.09 for $n = 60$.

7 Conclusion

In this paper, we considered the problem of minimum-energy scheduling for preemptive job sets, under the assumption that energy consumption is a convex function of processor speed. We first focus on the off-line scheduling of tree-structured job sets, where jobs are either properly nested or disjoint. Based on our observation that the optimal execution speeds form a non-decreasing sequence along any root-leaf path in the tree, we derived efficient bottom-up algorithm that computes the optimal voltage schedule for general tree-structured job sets. In addition, we gave an $O(n)$ algorithm for common-deadline or chain job sets.

We also studied the competitive ratio of on-line heuristic AVR for common-deadline job sets and limited-overlap job sets. A tight bound of 4 is proved in the former case, and an upper bound of 2.72 is proved in the latter case. Finally, we established that $\frac{17}{13}$ is a lower bound on the competitive ratio for any online schedule assuming that time is discrete.

Some interesting open problems remain. Our simulation results suggest that the number of critical intervals grows slowly with n; what exactly is the asymptotic rate? Can our findings for tree-structured job sets be generalized to other classes of job sets? Can the tree case itself be solved even more efficiently?

References

1. F. Yao, A. Demers and S. Shenker, *A Scheduling Model for Reduced CPU Energy*, Proceedings of the 36^{th} Annual Symposium on Foundations of Computer Science, 374-382, 1995.
2. Intel Corporation, *Wireless Intel SpeedStep Power Manager - Optimizing Power Consumption for the Intel PXA27x Processor Family*,Wireless Intel SpeedStep(R) Power Manager White Paper, 2004.
3. W. Kwon and T. Kim, *Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors*, 40^{th} Design Automation Conference, 2003.
4. B. Mochocki, X. S. Hu and G. Quan, *A Realistic Variable Voltage Scheduling Model for Real-Time Applications*, IEEE/ACM International Conference on Computer-Aided Design, 2002.

5. R. Jejurikar and R. K. Gupta, *Dynamic Voltage Scaling for Systemwide Energy Minimization in Real-Time Embedded Systems*, International Symposium on Low Power Electronics and Design, 2004.
6. H. S. Yun and J. Kim, *On Energy-Optimal Voltage Scheduling for Fixed-Priority Hard Real-Time Systems*, ACM Transactions on Embedded Computing Systems, 2(3): 393-430, 2003.
7. N. Bansal, T. Kimbrel and K. Pruhs, *Dynamic Speed Scaling to Manage Energy and Temperature*, Proceedings of the 45^{th} Annual Symposium on Foundations of Computer Science, 520-529, 2004.
8. M. Blum, R. Floyd, V. Pratt, R. Rivest and R. Tarjan, *Time Bounds for Selection*, Journal of Computer and System Sciences, 7:488-461, 1973.
9. J. Augustine, S. Irani and C. Swamy, *Optimal Power-Down Strategies*, Proceedings of the 45^th Annual Symposium on Foundations of Computer Science, 530-539, 2004.

Semi-online Problems on Identical Machines with Inexact Partial Information*

Zhiyi Tan and Yong He

Department of Mathematics, and State Key Lab of CAD & CG
Zhejiang University, Hangzhou 310027, P.R. China
{tanzy,mathhey}@zju.edu.cn

Abstract. In semi-online scheduling problems, we always assume that some partial additional information is exactly known in advance. This may not be true in some application. This paper considers semi-online problems on identical machines with inexact partial information. Three versions are considered, where we know in advance that the total size of all jobs, the optimal value, and the largest job size are in given intervals, respectively, while their exact values are unknown. We give both lower bounds of the problems and competitive ratios of algorithms as functions of a so-called disturbance parameter $r \in [1, \infty)$. We establish that for which r the inexact partial information is useful to improve the performance of a semi-online algorithm with respect to its pure online problem. Optimal or near optimal algorithms are then obtained.

1 Introduction

In scheduling theory, a problem is called *offline* if we have full information on the job data before constructing a schedule. If jobs arrive one by one and the jobs are required to be scheduled irrevocably on machines as soon as they are given, without any knowledge of the jobs that will arrive later, the problem is called *online*. If some partial additional information about the jobs is available in advance, and we cannot rearrange any job that has been assigned to machines, then the problem is called *semi-online*. Different partial information produces different semi-online problems. Algorithms for (semi-) online problems are called (*semi-*) *online algorithms*. Naturally, one wishes to achieve improvement of the performance of a semi-online algorithm with respect to its corresponding online problem by exploiting additional information. Though it is a relatively new area, various papers and a number of results on semi-online algorithms for scheduling problems have been published in the last decade. This paper will also consider the design and analysis of algorithms for semi-online scheduling.

In semi-online research, it is crucial to determine which type of partial information, and in how much extent, can improve the performance of a semi-online algorithm. Here we use the *competitive ratio* to measure the performance of an (semi-) online algorithm. For an (semi-) online algorithm A and a job sequence, let C^A denote the objective function value produced by A and let C^*

* Supported by NSFC (10301028, 10271110, 60021201) and TRAPOYT of China.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 297–307, 2005.

denote the optimal value in an offline version. Then the competitive ratio of A is $c = \inf\{c' : C^A \leq c'C^*, \text{ for any job sequence}\}$. An (semi-) online problem has a *lower bound* ρ if no (semi-) online algorithm has a competitive ratio of smaller than ρ. An (semi-) online algorithm is called *optimal* if its competitive ratio matches the lower bound. With these definitions, we say that an information is *useful* if it can admit an optimal semi-online algorithm with a competitive ratio smaller than that of an optimal online algorithm or the lower bound of the pure online problem.

We use the classical scheduling problem $P2||C_{\max}$ to illustrate the idea and method in this paper. Note that $P2||C_{\max}$ is the most commonly studied problem in the semi-online scheduling literature. It can be formulated as follows: We are given a sequence of independent jobs with sizes $p_1, p_2, \ldots, p_n$, which must be non-preemptively scheduled onto two identical machines M_1, M_2. We identify the jobs with their sizes. The objective is to minimize the maximum machine load (makespan), where the *load* of a machine is the total size of the jobs assigned to it. It is well known that algorithm LS is an optimal algorithm for the online version of $P2||C_{\max}$ with a competitive ratio of 3/2 [6], [5]. Here LS always assigns all the jobs to the machine with the smallest current load.

Several *basic semi-online* variants have been studied so far. Among others, Azar and Regev [1] considered the information that the optimal value C^* is known in advance (denoted by *opt*), which is also called the *online bin stretching problem*. Kellerer et al. [9] considered the information that the total size of all the jobs $T = \sum_{j=1}^{n} p_j$ is known in advance (denoted by *sum*), which can be also thought of the *generalized online bin stretching problem* [2]. He and Zhang [8] considered the information that the largest job size $p_{\max} = \max_{j=1,\cdots,n} p_j$ is known in advance (denoted by *max*). It is interesting that optimal algorithms for these semi-online problems have the identical competitive ratio of 4/3, which seems to imply that these types of partial information are of the same usefulness. Seiden et al. [10] studied another semi-online problem where jobs arrive in order of decreasing sizes (denoted by *decr*). They proved that LS is still optimal with a competitive ratio of 7/6.

To further shed light on usefulness of different types of information, *combined semi-online* problems have attracted researchers' attention. That is, determine whether a combination of two types of information can admit to construct a semi-online algorithm with much smaller competitive ratio than that of the case where only one type of information is available. Tan and He [11] pointed out several kinds of combination which make no sense. They further considered two combined semi-online problems. One is the version where both the information *sum* and *decr* are known in advance, and the other is the version where both *sum* and *max* are known in advance. Optimal algorithms with competitive ratios of 10/9 and 6/5 were provided, respectively. Epstein [4] considered a semi-online version with combined information *opt* and *decr*. An optimal algorithm with a competitive ratio of 10/9 was provided. Moreover, Dósa and He [3] studied the semi-online versions where one type of information and one type of additional algorithmic extension are combined.

In all the above considered semi-online problems, we assume that the known partial information is exact. However, this assumption may not be true in some application. Instead, we may know some partial information in advance, but this information is sensitive and not accurate, or with uncertainty. That is, we only know some *disturbed partial information* in advance. We would like to see whether it is still useful, and how to design an algorithm based on this inexact information if so. In this paper, we propose to introduce this concept in the context of semi-online scheduling. Three variants regarding the basic semi-online versions opt, sum, max will be studied. For the first variant $P2|dis\ opt|C_{\max}$, we know in advance that there exist some $p > 0$ and $r \geq 1$ such that $C^* \in [p, rp]$. For the second one $P2|dis\ sum|C_{\max}$, we know in advance that there exist some $p > 0$ and $r \geq 1$ such that $T \in [p, rp]$. For the last one $P2|dis\ max|C_{\max}$, we know in advance that there exist some $p > 0$ and $r \geq 1$ such that $p_{\max} \in [p, rp]$. We call r the *disturbance parameter*. We will present the competitive ratios of semi-online algorithms and lower bound as functions of the r for each problem. The results are summarized in Table 1, and Fig. 1 further depicts the competitive ratios and the lower bounds of the three problems.

Table 1. The obtained results in this paper.

problem	*dis opt*	*dis sum*	*dis max*
the interval of r where the obtained algorithm is optimal	$[\frac{3+\sqrt{21}}{6}, \frac{1+\sqrt{10}}{3}]$ $\cup[\frac{6-\sqrt{10}}{2}, \infty)$	$[\frac{3+\sqrt{21}}{6}, \infty)$	$[1, \sqrt{5}-1]$ $\cup[2, \infty)$
the gap between the upper and lower bound	≤ 0.0303	≤ 0.0303	≤ 0.0445
the total length of non-optimal interval	≤ 0.2957	≤ 0.2635	≤ 0.7640

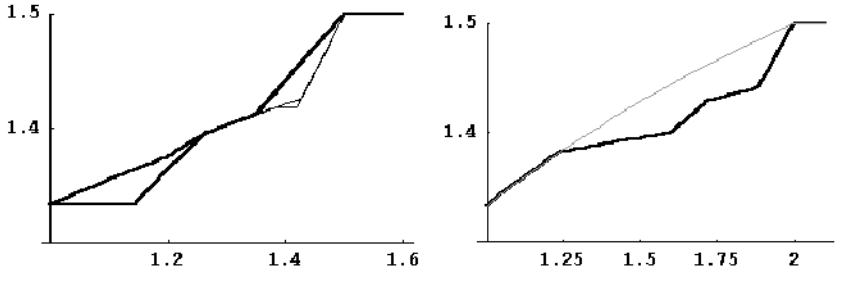


Fig. 1. Left: The competitive ratios and lower bounds of $P2|dis\ sum|C_{\max}$ and $P2|dis\ opt|C_{\max}$. Solid curves are for the former problem and dashed curves for the latter problem. Right: The competitive ratio and lower bound of $P2|dis\ max|C_{\max}$.

As the obtained competitive ratios and lower bounds are functions of the disturbance parameter, we can see in what extent the disturbed information is useful. For example, since the lower bounds of the first two problems are $3/2$ when $r \geq 3/2$, we conclude that the disturbed information becomes useless. When $1 \leq r < 3/2$, each type of disturbed information is still useful, and we

can know how the disturbance parameter affects the competitive ratios/lower bounds from the obtained parametric competitive ratios/lower bounds.

Moreover, although *sum* is a relaxation of *opt*, all the results resemble or even identical in the literature (e.g., competitive ratios, algorithms, and lower bounds) for $P|sum|C_{\max}$ and $P|opt|C_{\max}$. That is to say, an algorithm designed for one problem may be applied to another with the same competitive ratio, and they have the same lower bound [2]. However, we will show that this may not be valid for our problems. In fact, when $1.3502 \leq r \leq 3/2$, the lower bound for $P2|dis\ opt|C_{\max}$ is smaller than that for $P2|dis\ sum|C_{\max}$, and our algorithm $H1$ also has a smaller competitive ratio when it is applied to $P2|dis\ opt|C_{\max}$. Therefore, these two types of information have some similarity in semi-online algorithm design and analysis, but definitely are not identical.

A little bit closely related problem is the basic semi-online problem of $P2||C_{\max}$ where it is assumed that there exist some $p > 0$ and $r \geq 1$ such that all the job sizes are in $[p, rp]$. He and Zhang [8] proved LS is an optimal algorithm with a competitive ratio of $(1 + \min\{r, 2\})/2$. He and Dósa [7] further considered it on three identical machines. However, this problem is different from those considered in this paper, since for this problem, we know the exact information of every job once it arrives (for our considered problems we never know the exact information of *opt*, *sum* and *max*, respectively). In addition, we know in advance that every size is between a given upper and lower bound. Hence, it is essentially semi-online with a type of exact information. Moreover, this problem becomes optimally solvable for the case of $r = 1$ (we then have full information of job sequence, and it is a very special case that all jobs are identical), which is also unlike that in our considered problems, they become semi-online ones $P2|opt|C_{\max}$, $P2|sum|C_{\max}$ and $P2|max|C_{max}$ when $r = 1$.

The paper is organized as follows. Section 2 considers $P2|dis\ opt|C_{\max}$ and $P2|dis\ sum|C_{\max}$ simultaneously. Section 3 considers $P2|dis\ \ max|C_{\max}$. When analyzing a semi-online algorithm, we denote by l_i^t the current load of M_i right before the assignment of p_t, and l_i the final load of M_i after all the jobs have been assigned, $i = 1, 2$.

2 Problems with *dis opt* and *dis sum*

By normalization, we assume that $p = 1$ when considering $P2|dis\ opt|C_{\max}$, and $p = 2$ when considering $P2|dis\ sum|C_{\max}$.

2.1 Lower Bounds

Theorem 1. *Any semi-online algorithm A for* $P2|dis\ opt|C_{\max}$ *has a competitive ratio of at least*

$$\begin{cases} \frac{4}{3}, & for\ \ 1 \leq r \leq \frac{8}{7} \approx 1.14286, \\ \frac{5r}{2r+2}, & for\ \ \frac{8}{7} \leq r \leq \frac{3+\sqrt{21}}{6} \approx 1.26376, \\ \frac{7r+1}{4r+2}, & for\ \ \frac{3+\sqrt{21}}{6} \leq r \leq \frac{1+\sqrt{10}}{3} \approx 1.38743, \\ \frac{6-\sqrt{10}}{2}, & for\ \ \frac{1+\sqrt{10}}{3} \leq r \leq \frac{6-\sqrt{10}}{2} \approx 1.41886, \\ r, & for\ \ \frac{6-\sqrt{10}}{2} \leq r \leq \frac{3}{2}, \\ \frac{3}{2}, & for\ \ r \geq \frac{3}{2}. \end{cases}$$

Proof. Since 4/3 is the lower bound for the problem $P2|opt|C_{\max}$, it is still valid for $P2|dis\ opt|C_{\max}$ when $1 \le r \le 8/7$.

Remember that $C^* \in [1, r]$ in the following proof. Assume $\frac{3+\sqrt{21}}{6} \le r \le \frac{1+\sqrt{10}}{3}$. Let $p_1 = \frac{2r^2-r-1}{4r+2}$ and $p_2 = \frac{3r^2-r}{4r+2}$. If they are assigned to the same machine, for example, M_1, let $p_3 = \frac{r^2+3r}{4r+2}$. If p_3 is also assigned to M_1, let the last two jobs be $p_4 = \frac{3+6r-5r^2}{4r+2}$ and $p_5 = \frac{2+r-r^2}{4r+2}$. Then we have $C^A \ge p_1+p_2+p_3 = \frac{6r^2+r-1}{4r+2} \ge \frac{7r+1}{4r+2}$ while $C^* = 1$. It follows that $\frac{C^A}{C^*} \ge \frac{7r+1}{4r+2}$. If p_3 is assigned to M_2, let $p_4 = \frac{2r^2+3r+1}{4r+2}$. We obtain $C^A \ge \min\{p_1+p_2+p_4, p_3+p_4\} = \min\{\frac{7r^2+r}{4r+2}, \frac{3r^2+6r+1}{4r+2}\} = \frac{7r^2+r}{4r+2}$, and $C^* = r$, resulting in $\frac{C^A}{C^*} \ge \frac{7r+1}{4r+2}$.

Hence, we only need to consider the case that A assigns p_i to M_i, $i = 1, 2$. Then let $p_3 = \frac{4r-2r^2}{4r+2}$. If p_3 is assigned to M_1, let the last two jobs be $p_4 = 1$ and $p_5 = \frac{3+2r-3r^2}{4r+2}$. We have $C^* = 1$ and $C^A \ge \min\{p_1+p_3+p_4, p_2+p_4\} = \min\{\frac{7r+1}{4r+2}, \frac{3r^2+3r+2}{4r+2}\} = \frac{7r+1}{4r+2}$. It follows that $\frac{C^A}{C^*} \ge \frac{7r+1}{4r+2}$. If p_3 is assigned to M_2, let $p_4 = \frac{1+4r-r^2}{4r+2}$. If p_4 is assigned to M_1, let the last job be $p_5 = \frac{6r^2-2r}{4r+2}$. We have $C^A \ge \min\{p_1+p_4+p_5,\ p_2+p_3+p_5\} = \frac{7r^2+r}{4r+2}$, and $C^* = r$, implying $\frac{C^A}{C^*} \ge \frac{7r+1}{4r+2}$. If p_4 is assigned to M_2, let the last two jobs be $p_5 = p_6 = \frac{2+r-r^2}{4r+2}$. We have $C^A \ge p_2+p_3+p_4 = \frac{7r+1}{4r+2}$, and $C^* = 1$. It follows that $\frac{C^A}{C^*} \ge \frac{7r+1}{4r+2}$, too.

The result for the remaining cases can be shown similarly. □

Let $r_1 \approx 1.3502$ be the solution of the equation $\frac{7r+1}{4r+2} = \frac{\sqrt{1+32r}-1}{4}$. The following theorem states that the *dis sum* problem admits a greater lower bound for $r_1 \le r \le 3/2$.

Theorem 2. *Any algorithm A for $P2|dis\ sum|C_{\max}$ has a competitive ratio of at least*

$$\begin{cases} \frac{4}{3}, & for\ \ 1 \le r \le \frac{8}{7}, \\ \frac{5r}{2r+2}, & for\ \ \frac{8}{7} \le r \le \frac{3+\sqrt{21}}{6}, \\ \frac{7r+1}{4r+2}, & for\ \ \frac{3+\sqrt{21}}{6} \le r \le r_1, \\ \frac{\sqrt{1+32r}-1}{4}, & for\ \ r_1 \le r \le \frac{3}{2}, \\ \frac{3}{2}, & for\ \ r \ge \frac{3}{2}. \end{cases}$$

2.2 Description of Algorithms

In this subsection, we present our semi-online algorithms. Noting that the lower bounds are 3/2 for any $r \ge 3/2$, LS is optimal for the problems considered in this section. That is to say, the disturbed information becomes useless. Hence, we focus on the interval of $r \in [1, 3/2]$ in the following. Let c be a parameter satisfying $r \le c \le 2$ whose exact value will be specified later. In fact, c will be the desired competitive ratios of our algorithms. We call the process that assigns jobs one by one by an algorithm as a *scheduling process.*

Definition 1. *(1) If right before the assignment of job p_t, $l^t_{i_1} \in [(2-c)r,\ c]$ and $l^t_{i_2} \le c$ hold, where $\{i_1, i_2\} = \{1, 2\}$, then we say that a scheduling process is in Stopping Condition 1 (SC1 for short).*

(2) If right before the assignment of job p_t, $l_{i_1}^t \in [(4-2c)r-c,\ c-1]$ and $l_{i_2}^t < (2-c)r$ hold, where $\{i_1, i_2\} = \{1, 2\}$, then we say that a scheduling process is in Stopping Condition 2 (SC2 for short).

We further present two basic assignment procedures as follows.

Assignment Procedure 1 (*AP*1 for short): Assign p_t and all the remaining jobs to M_{i_2}. Stop.

Assignment Procedure 2 (*AP*2 for short): Assign p_t and subsequent arriving jobs to M_{i_2} until there exists a job $p_{t'}$ such that $l_{i_2}^{t'} < (2-c)r$ and $l_{i_2}^{t'} + p_{t'} \geq (2-c)r$. We do

1. If $l_{i_2}^{t'} + p_{t'} \in [(2-c)r,\ c]$, assign $p_{t'}$ to M_{i_2} and all the remaining jobs according to $AP1$. Stop.
2. If $l_{i_2}^{t'} + p_{t'} > c$ and $l_{i_1}^{t'} + p_{t'} \leq c$, assign $p_{t'}$ to M_{i_1} and all the remaining jobs according to $AP1$. Stop.
3. If $l_{i_2}^{t'} + p_{t'} > c$ and $l_{i_1}^{t'} + p_{t'} > c$, assign $p_{t'}$ to M_{i_1} and all the remaining jobs to M_{i_2}. Stop.

Lemma 1. *For both problems considered in this section, if the scheduling process of an algorithm A is in SC1 (SC2) right before assigning p_t, then AP1 (AP2) results in $C^A/C^* \leq c$.*

The main idea of the algorithms can be stated as follows. When we schedule the jobs we try to achieve SC1 or SC2 of the scheduling process as early as possible. If it is fulfilled, we then assign all the remaining jobs according to corresponding assignment procedures $AP1$ or $AP2$, which guarantees the desired competitive ratio by Lemma 1. If the scheduling process cannot be in one of $SC1$ and $SC2$, then there must exist some jobs with larger sizes (e.g., $p_s, s \in \{a, b, d, e, f\}$ in the algorithm description) that prevent the scheduling process from $SC1$ or $SC2$. Since we have properly assigned these jobs, the competitive ratio remains valid. Note that for the former case, the lower bound 1 of the C^* is sufficient for obtaining the desired competitive ratio, whereas for the latter case, we need to establish tighter estimate of the C^*.

Algorithm *H*1:

1. Assign jobs to M_1 until there exists a job p_a such that $l_1^a < (4-2c)r-c$ and $l_1^a + p_a \geq (4-2c)r-c$.
   (1.1) If $l_1^a + p_a \in [(4-2c)r-c,\ c-1]$, assign p_a to M_1 and all the remaining jobs by $AP2$. Stop.
   (1.2) If $l_1^a + p_a \in (c-1,\ (2-c)r)$, assign p_a to M_2.
      (1.2.1) if $p_a \leq c-1$, assign all the remaining jobs by $AP2$. Stop.
      (1.2.2) if $p_a > c-1$, go to Step 2.
   (1.3) If $l_1^a + p_a \in [(2-c)r,\ c]$, assign p_a to M_1 and all the remaining jobs by $AP1$. Stop.
   (1.4) If $l_1^a + p_a > c$, assign p_a to M_1 and all the remaining jobs to M_2. Stop.
2. Assign subsequent arriving jobs to M_1 until there exists a job p_b such that $l_1^b < (4-2c)r-c$ and $l_1^b + p_b \geq (4-2c)r-c$.

(2.1) If $l_1^b + p_b \in [(4-2c)r - c,\ c-1]$, assign p_b to M_1 and all the remaining jobs by $AP2$. Stop.
(2.2) If $l_1^b + p_b \in (c-1,\ (2-c)r)$, and
(2.2.1) if $p_a + p_b < (2-c)r$, assign p_b to M_2 and go to Step 3.
(2.2.2) if $p_a + p_b \in [(2-c)r,\ c]$, assign p_b to M_2 and all the remaining jobs by $AP1$. Stop.
(2.2.3) if $p_a + p_b > c$, assign p_b to M_1 and go to Step 5.
(2.3) If $l_1^b + p_b \in [(2-c)r,\ c]$, assign p_b to M_1 and all the remaining jobs by $AP1$. Stop.
(2.4) If $l_1^b + p_b > c$, assign p_b to M_1 and all the remaining jobs to M_2. Stop.

3. Assign subsequent arriving jobs to M_1 until there exists a job p_d such that $l_1^d < (4-2c)r - c$ and $l_1^d + p_d \geq (4-2c)r - c$.
(3.1) If $l_1^d + p_d \in [(4-2c)r - c,\ c-1]$, assign p_d to M_1 and all the remaining jobs by $AP2$. Stop.
(3.2) If $l_1^d + p_d \in (c-1,\ (2-c)r)$, and
(3.2.1) if $p_a + p_b + p_d \leq c$, assign p_d to M_2 and all the remaining jobs by $AP1$. Stop.
(3.2.2) if $p_a + p_b + p_d > c$, assign p_d to M_1 and go to Step 4.
(3.3) If $l_1^d + p_d \in [(2-c)r,\ c]$, assign p_d to M_1 and all the remaining jobs by $AP1$. Stop.
(3.4) If $l_1^d + p_d > c$, assign p_d to M_1 and all the remaining jobs to M_2. Stop.

4. Assign subsequent arriving jobs to M_1 until there exists a job p_e such that $l_1^e < (2-c)r$ and $l_1^e + p_e \geq (2-c)r$.
(4.1) If $l_1^e + p_e \in [(2-c)r,\ c]$, assign p_e to M_1 and all the remaining jobs by $AP1$. Stop.
(4.2) If $l_1^e + p_e > c$, and
(4.2.1) if $p_a + p_b + p_e \leq c$, assign p_e to M_2 and all the remaining jobs by $AP1$. Stop.
(4.2.2) if $p_a + p_b + p_e > c$, assign p_e to a machine by LS algorithm, and all the remaining jobs to another machine. Stop.

5. Assign subsequent arriving jobs to M_1 until there exists a job p_f such that $l_1^f < (2-c)r$ and $l_1^f + p_f \geq (2-c)r$.
(5.1) If $l_1^f + p_f \in [(2-c)r,\ c]$, assign p_f to M_1 and all the remaining jobs by $AP1$. Stop.
(5.2) If $l_1^f + p_f > c$, and
(5.2.1) if $p_a + p_f \leq c$, assign p_f to M_2 and all the remaining jobs by $AP1$. Stop.
(5.2.2) if $p_a + p_f > c$, assign p_f to a machine by LS algorithm, and all the remaining jobs to another machine. Stop.

Algorithm $H2$:

1. Assign jobs to M_1 until there exists a job p_a such that $l_1^a < (4-2c)r - c$ and $l_1^a + p_a \geq (4-2c)r - c$.
(1.1) If $l_1^a + p_a \in [(4-2c)r - c,\ c-1]$, assign p_a to M_1 and all the remaining jobs by $AP2$. Stop.

(1.2) If $l_1^a + p_a \in (c-1,\ 6r-2c+1-3cr)$, assign p_a to M_1 and go to Step 2.
(1.3) If $l_1^a + p_a \in [6r-2c+1-3cr,\ c(1+r)-2r]$, assign p_a to M_1 and go to Step 3.
(1.4) If $l_1^a + p_a \in (c(1+r)-2r,\ (2-c)r)$, assign p_a to M_2 and go to Step 4.
(1.5) If $l_1^a + p_a \in [(2-c)r,\ c]$, assign p_a to M_1 and all the remaining jobs by $AP1$. Stop.
(1.6) If $l_1^a + p_a > c$, assign p_a to M_1 and all the remaining jobs to M_2. Stop.

2. Assign subsequent arriving jobs to M_1 until there exists a job p_b such that $l_1^b < 6r-2c+1-3cr$ and $l_1^b + p_b \geq 6r-2c+1-3cr$.
(2.1) If $l_1^b + p_b \in [6r-2c+1-3cr,\ c(1+r)-2r]$, assign p_b to M_1 and go to Step 3.
(2.2) If $l_1^b + p_b \in (c(1+r)-2r,\ (2-c)r)$, assign p_b to M_2 and all the remaining jobs by $AP2$. Stop.
(2.3) If $l_1^b + p_b \in [(2-c)r,\ c]$, assign p_b to M_1 and all the remaining jobs by $AP1$. Stop.
(2.4) If $l_1^b + p_b > c$, assign p_b to M_2.
  (2.4.1) if $p_b \leq c$, assign all the remaining jobs by $AP1$. Stop.
  (2.4.2) if $p_b > c$, assign all the remaining jobs to M_1. Stop.

3. Assign subsequent arriving jobs to M_2 until there exists a job p_d such that $l_2^d < (4-2c)r-c$ and $l_2^d + p_d \geq (4-2c)r-c$.
(3.1) If $l_2^d + p_d \in [(4-2c)r-c,\ c-1]$, assign p_d to M_2 and all the remaining jobs by $AP2$. Stop.
(3.2) If $l_2^d + p_d \in (c-1,\ (2-c)r)$, assign p_d to M_1 and all the remaining jobs by $AP1$. Stop.
(3.3) If $l_2^d + p_d \in [(2-c)r,\ c]$, assign p_d to M_2 and all the remaining jobs by $AP1$. Stop.
(3.4) If $l_2^d + p_d > c$, assign p_d to M_2 and all the remaining jobs to M_1. Stop.

4. Assign subsequent arriving jobs to M_1 until there exists a job p_e such that $l_1^e < (4-2c)r-c$ and $l_1^e + p_e \geq (4-2c)r-c$.
(4.1) If $l_1^e + p_e \in [(4-2c)r-c,\ c-1]$, assign p_e to M_1 and all the remaining jobs by $AP2$. Stop.
(4.2) If $l_1^e + p_e \in (c-1,\ (2-c)r)$, and
  (4.2.1) if $p_a + p_e \leq c$, assign p_e to M_2 and all the remaining jobs by $AP1$. Stop.
  (4.2.2) if $p_a + p_e > c$, assign p_e to M_1 and go to Step 5.
(4.3) If $l_1^e + p_e \in [(2-c)r,\ c]$, assign p_e to M_1 and all the remaining jobs by $AP1$. Stop.
(4.4) If $l_1^e + p_e > c$, assign p_e to M_1 and all the remaining jobs to M_2. Stop.

5. Assign subsequent arriving jobs to M_1 until there exists a job p_f such that $l_1^f < (2-c)r$ and $l_1^f + p_f \geq (2-c)r$.
(5.1) If $l_1^f + p_f \in [(2-c)r,\ c]$, assign p_f to M_1 and all the remaining jobs by $AP1$. Stop.
(5.2) If $l_1^f + p_f > c$, and
  (5.2.1) if $p_a + p_f \leq c$, assign p_f to M_2 and all the remaining jobs by $AP1$. Stop.
  (5.2.2) if $p_a + p_f > c$, assign p_f to a machine by LS algorithm, and all the remaining jobs to another machine. Stop.

2.3 Competitive Analysis

Theorem 3. *The competitive ratio of $H1$ for $P2|dis\ sum|C_{\max}$ is*

$$c = \max\{\frac{7r+1}{4r+2}, \frac{\sqrt{1+32r}-1}{4}\} = \begin{cases} \frac{7r+1}{4r+2}, & for \quad 1 \le r < r_1, \\ \frac{\sqrt{1+32r}-1}{4}, & for \quad r_1 \le r < \frac{3}{2}. \end{cases}$$

Proof. It is easy to verify that for $1 \le r \le 3/2$, the value of c satisfies $0 < (4-2c)r - c < c-1 < (2-r)c < c$. So, algorithm $H1$ is well-defined. To obtain the desired competitive ratio, we distinguish three cases with regard to how $H1$ terminates.

Case 1 The scheduling process stops at one of Steps 1.1, 2.1, 3.1, 2.2.2, 4.1, 5.1, 1.3, 2.3 and 3.3. Then from the description of $H1$, we know that the scheduling process has been in $SC1$ or $SC2$ right before entering any one of these steps. Hence, the desired competitive ratio follows directly from Lemma 1.

Case 2 The scheduling process stops at one of Steps 1.2.1, 3.2.1, 4.2.1 and 5.2.1. We show one by one that the scheduling process has indeed been in $SC1$ or $SC2$ before stopping, too. We only prove the result for the subcase of Step 1.2.1, other subcases can be done similarly.

For Step 1.2.1, according to the conditions of Steps 1 and 1.2, we have $l_1^a < (4-2c)r-c$ and $l_1^a + p_a > c-1$, i.e. $p_a > (c-1)-((4-2c)r-c) = 2c-1-(4-2c)r$. Note that if $1 \le r \le \frac{5+\sqrt{41}}{8}$, we have $c \ge \frac{7r+1}{4r+2} > \frac{8r+1}{4r+3}$; and if $\frac{5+\sqrt{41}}{8} \le r \le \frac{3}{2}$, we have $c \ge r > \frac{8r+1}{4r+3}$. Therefore, $p_a > 2c-1-(4-2c)r > (4-2c)r-c$. On the other hand, the condition of Step 1.2.1 states $p_a \le c-1$. Hence, $p_a \in [(4-2c)r-c,\ c-1]$. Since p_a is the first job assigned to M_2 and $l_1^a \le l_1^a + p_a < (2-c)r$, the scheduling process is in $SC2$ right after assigning p_a to M_2.

Case 3 The scheduling process stops at one of all the remaining Steps 1.4, 2.4, 3.4, 4.2.2 and 5.2.2. We show that the desired competitive ratio is still valid. We first claim that it is impossible that the final loads of the two machines are greater than c. In fact, if $l_1 > c$ and $l_2 > c$, then $T = l_1 + l_2 > 2c \ge 2r$, contradicting $T \in [2, 2r]$. Therefore, if there exist an $i_1 \in \{1, 2\}$ such that $l_{i_1} > c$, then $C^{H1} = l_{i_1}$. Hence, if the scheduling process stops at one of Steps 1.4, 2.4 and 3.4, we always have $C^{H1} = l_1^s + p_s$, where $s \in \{a, b, d\}$. Combining it with $l_1^s < (4-2c)r-c$ and $l_1^s + p_s > c$, we establish $\frac{C^{H1}}{C^*} \le \frac{l_1^s+p_s}{p_s} \le 1 + \frac{l_1^s}{p_s} \le 1 + \frac{l_1^s}{c-l_1^s} = \frac{c}{c-l_1^s} \le \frac{c}{c-((4-2c)r-c)} \le c$, where the last inequality is because $c-((4-2c)r-c) \ge 1$ (due to $c \ge \frac{7r+1}{4r+2} > \frac{4r+1}{2r+2}$). Hence, we are left to consider the subcases that the scheduling process stops at Steps 4.2.2 and 5.2.2, which will be done by contradiction. Hence, we suppose $C^{H1}/C^* \ge c$.

Noting that for these two subcases, assigning the job p_s, $s = e, f$, to the machine with current smaller load still makes its completion time greater than c, so we have $C^{H1} = \min\{l_1^s + p_s, l_2^s + p_s\} > c$ and $C^{H1} \le (T - p_s)/2 + p_s = (T + p_s)/2$. We only prove the result for the subcase of Step 5.2.2, the subcase of Step 4.2.2 can be done similarly. For Step 5.2.2, we have $C^{H1} \le \min\{(T + p_f)/2,\ p_a + p_f\}$. Consider the assignment of p_a, p_b, p_f in the optimal schedule. Three subcases are considered as follows.

(I) p_a and p_f are assigned to the same machine. Then $C^* \geq p_a + p_f \geq C^{H1}$, a contradiction. (II) p_a and p_b are assigned to the same machine. Then $C^* \geq p_a+p_b$. Substituting it and $C^{H1} \leq (T+p_f)/2$ into $C^{H1}/C^* \geq c$, we have $T+p_f > 2c(p_a+p_b)$. Since before entering Step 5.2.2, the scheduling process have entered Step 2.2.3, and thus $p_a+p_b > c$. Therefore, $2T \geq T+p_a+p_b+p_f > 2c^2+c \geq 4r$, where the last inequality is due to $c \geq \frac{\sqrt{1+32r}-1}{4}$. It contradicts $T \in [2,\ 2r]$. (III) p_b and p_f are assigned to the same machine. Then $C^* \geq p_b + p_f$. Substituting it and $C^{H1} \leq p_a + p_f$ into $C^{H1}/C^* \geq c$, we obtain $p_a \geq cp_b + (c-1)p_f$. Noting that $p_a + p_b > c$ and $p_a + p_f > c$ (due to the condition of Steps 2.2.3 and 5.2.2), we have $2cp_a \geq c(p_a+p_b) + (c-1)(p_a+p_f) > (2c-1)c$. It follows that $p_a > (2c-1)/2 > (2-c)r$, contradicting the condition of Step 1.2.

In summary, we have shown that for all possible cases $C^{H1}/C^* \leq c$ holds. □

Theorem 4. *The competitive ratio of $H1$ for $P2|dis\ opt|C_{\max}$ is*

$$c = \max\{\frac{7r+1}{4r+2},\ r\} = \begin{cases} \frac{7r+1}{4r+2}, & for\ 1 \leq r < \frac{5+\sqrt{41}}{8}, \\ r, & for\ \frac{5+\sqrt{41}}{8} \leq r < \frac{3}{2}. \end{cases}$$

For $1 \leq r \leq \frac{3+\sqrt{21}}{6}$, $H2$ can be better than $H1$ for both two problems.

Theorem 5. *If $1 \leq r \leq \frac{3+\sqrt{21}}{6}$, then for $P2|dis\ sum|C_{\max}$ and $P2|dis\ \ opt|C_{\max}$ algorithm $H2$ has a competitive ratio of*

$$c = \max\{\frac{2r+2}{r+2},\ \frac{12r+1}{6r+4}\} = \begin{cases} \frac{2r+2}{r+2}, & for\ \ 1 \leq r < \frac{6}{5}, \\ \frac{12r+1}{6r+4}, & for\ \ \frac{6}{5} \leq r \leq \frac{3+\sqrt{21}}{6}. \end{cases}$$

3 Problem with *dis max*

By normalization, we assume that $p = 1$ in this section. Hence $p_{\max} \in [1, r]$.

Theorem 6. *Any semi-online algorithm A for $P2|dis\ max|C_{\max}$ has a competitive ratio of at least*

$$\begin{cases} \frac{2r+2}{r+2}, & for\ \ 1 \leq r \leq \sqrt{5}-1 \approx 1.2360, \\ \frac{\sqrt{r^2+2r}-(r-2)}{2}, & for\ \ \sqrt{5}-1 \leq r \leq \frac{8}{5} = 1.6000, \\ \frac{r+4}{4}, & for\ \ \frac{8}{5} \leq r \leq \frac{12}{7} \approx 1.7143, \\ \frac{\sqrt{9r^2+28r+4}-(3r-2)}{4}, & for\ \ \frac{12}{7} \leq r \leq \frac{7+\sqrt{65}}{8} \approx 1.8828, \\ \frac{r+1}{2}, & for\ \ \frac{7+\sqrt{65}}{8} \leq r \leq 2, \\ \frac{3}{2}, & for\ \ r \geq 2. \end{cases}$$

Because the lower bound is 3/2 for any $r \geq 2$, we focus on the interval of $r \in [1, 2]$. The following algorithm is modified from PLS [8], which is optimal for $P2|max|C_{\max}$.

Algorithm $MPLS$:

1. Assign jobs to M_1 until there exists a job p_a such that one of the following conditions happens: (1.1) $p_a \in [1, r]$; (1.2) $l_1^a + p_a > 2$.
2. Assign p_a to M_2, and all the remaining jobs by LS algorithm.

Theorem 7. *For $P2|dis\ max|C_{\max}$, algorithm $MPLS$ has a competitive ratio of $\frac{2r+2}{r+2}$ for $r \in [1, 2]$, and is optimal for $r \in [1, \sqrt{5} - 1]$.*

References

1. Y. Azar, O. Regev, On-line bin-stretching, *Theoretical Computer Science*, **168,** 17-41(2001).
2. T. C. E. Cheng, H. Kellerer, V. Kotov, Semi-on-line multiprocessor scheduling with given total processing time, *Theoretical Computer Science*. Published online doi:10.1016/j.tcs.2004.11.018
3. G. Dósa, Y. He, Semi-online agorithms for parallel machine scheduling problems, *Computing*, **72,** 355-363(2004).
4. L. Epstein, Bin stretching revisited, *Acta Informatica*, **39,** 97-117(2003).
5. U. Faigle, W. Kern, G. Turan, On the performance of online algorithms for partition problems, *Acta Cybernetica*, **9,** 107-119(1989).
6. R. L. Graham, Bounds for certain multiprocessor anomalies, *Bell Systems Technical Journal*, **45,** 1563-1581(1966).
7. Y. He, G. Dósa, Semi-online scheduling jobs with tightly-grouped processing times on three identical machines, *Discrete Applied Mathematics*, to appear.
8. Y. He, G. Zhang, Semi on-line scheduling on two identical machines. *Computing,* **62,** 179-187(1999).
9. H. Kellerer, V. Kotov, M. G. Speranza, Z. Tuza, Semi on-line algorithms for the partition problem, *Operations Research Letters,* **21,** 235-242(1997).
10. S. Seiden, J. Sgall, G. Woeginger, Semi-online scheduling with decreasing job sizes, *Operations Research Letters,* **27,** 215-227(2000).
11. Z. Y. Tan, Y. He, Semi-on-line problems on two identical machines with combined partial information, *Operations Research Letters,* **30,** 408-414(2002).

On-Line Simultaneous Maximization of the Size and the Weight for Degradable Intervals Schedules

Fabien Baille, Evripidis Bampis, Christian Laforest, and Nicolas Thibault

Tour Evry 2, LaMI, Université d'Evry, 523 place des terrasses, 91000 EVRY France
{fbaille,bampis,laforest,nthibaul}@lami.univ-evry.fr

Abstract. We consider the problem of scheduling *on-line* a sequence of *degradable* intervals in a set of k identical machines. Our objective is to find a schedule that maximizes *simultaneously* the *Weight* (equal to the sum of processing times) and the *Size* (equal to the number) of the scheduled intervals. We propose a *bicriteria* algorithm that uses the strategies of two monocriteria algorithms (GOL [7], maximizing the *Size* and LR [4], maximizing the *Weight*) and yields two simultaneous constant competitive ratios. This work is an extension of [2] (COCOON'04), where the same model of degradable intervals was investigated in an *off-line* context and the two objectives were considered separately.

In this paper, we consider the problem of scheduling on-line degradable intervals on k identical machines. We define a *degradable* interval σ by a triplet (r, q, d) where r denotes the *release date*, q the *minimal deadline* and d the *deadline* $(r < q \leq d)$. This means that σ is *scheduled* if and only if it is executed from date r to date t $(q \leq t \leq d)$ on one machine. Intuitively, in this model, each interval can be shortened (with respect to the required total execution $[r, d)$). We denote by $[r, t)$ the numerical interval corresponding to the effective execution of a degradable interval σ and by $p(\sigma) = t - r$ its processing time. We define the *weight* $w(\sigma)$ of the effective execution of any interval σ by $w(\sigma) = t_\sigma - r_\sigma$. This means that the weight of an interval σ is equal to its processing time (it is known in the literature as the *proportional weight* model [8]). In our model, we consider on-line sequences of degradable intervals $\sigma_1, ..., \sigma_n$ where the σ_i's are *revealed* one by one in the increasing order of their release dates $(r_1 \leq r_2 \leq \cdots \leq r_n)$, and future intervals are not known in advance.

For any algorithm A, we denote by A^k the version of A running on k identical machines. In our model, an on-line algorithm A^k has to build at each step a *valid* schedule. A schedule is *valid* if and only if for every date t, there is at most one interval on each machine and each interval is scheduled at most once. When a new interval σ_i is revealed (at step i), the algorithm A^k can *reject* it (in this case, it is definitively lost) or *serve* it. In this second case, if the algorithm schedules σ_i on machine j, it interrupts at least the already scheduled interval intersecting σ_i on machine j. The interrupted intervals are definitively lost and no gain is obtained from them for both metrics. Thus, each step i of any on-line algorithm

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 308–317, 2005.

A^k can be decomposed into two stages: First, there is the *interrupting stage* of step i. During this stage, the algorithm interrupts a subset of the already scheduled intervals (note that this subset can be empty). Secondly, there is the *scheduling stage* of step i. During this stage, the algorithm decides if the new interval σ_i is served or rejected, and if it is served, on which machine σ_i is served.

Notation 1 (Schedule $A^k(\sigma_1, \ldots, \sigma_i)$) *Let $\sigma_1, \ldots, \sigma_n$ be any on-line sequence of degradable intervals and let A^k be any algorithm running on k identical machines. For every step i ($1 \leq i \leq n$), we denote by $A^k(\sigma_1, \ldots, \sigma_i)$ the schedule returned by A^k at the end of step i.*

We define the *size* $N(O) = |\{\sigma \in O\}|$ (i.e. the number of scheduled intervals) and the *weight* $W(O) = \sum_{\sigma \in O} w(\sigma)$ (i.e. the weight of scheduled intervals) of any schedule O. Our problem is then to find a schedule which has *size* and *weight* the largest possible. In order to evaluate the quality of a schedule for our measures (the Size *and* the Weight), we use the competitive ratio [5].

Definition 1 (Competitive ratio). *Let $\sigma_1, ..., \sigma_n$ be any on-line sequence of intervals. Let $A^k(\sigma_1, ..., \sigma_i)$ be the schedule on k machines given by an algorithm A^k at step i ($1 \leq i \leq n$) and let O_i^* be the optimal (off-line) schedule on k machines of $\{\sigma_1, \ldots, \sigma_i\}$ for the criterion C (here, $C = W$ or $C = N$). A^k has a competitive ratio of ρ (it is ρ-competitive) for the criterion C if and only if we have:*

$$\forall i,\ 1 \leq i \leq n,\ \rho \cdots C(A^k(\sigma_1, ..., \sigma_i)) \geq C(O_i^*)$$

An algorithm A^k is (ρ, μ)-competitive if it is *simultaneously* ρ-competitive for the Size and μ-competitive for the Weight. In this paper, we propose a $\left(\frac{k}{r}, \frac{4k}{k-r-2}\right)$-competitive algorithm, called AB^k (with $1 \leq r < k$). For example, if we set $r = \frac{k}{2}$ (if $\frac{k}{2} \geq 3$ and k even), AB^k is $\left(2, \frac{8}{1-\frac{4}{k}}\right)$-competitive.

Previous Works. The off-line version of the bicriteria non-degradable problem has been treated in [3] where a $(\frac{k}{r}, \frac{k}{k-r})$-approximation algorithm ($1 \leq r < k$) has been proposed. Concerning the monocriteria non-degradable problems, they have been extensively studied for both the off-line and the on-line versions. In particular, the off-line versions are polynomial (see Faigle and Nawijn [7] for the Size and Carlisle and Lloyd [6] or Arkin and Silverberg [1] for the Weight problems). In the on-line context, the algorithm GOL of Faigle and Nawijn [7] is optimal for the Size problem. For the Weight problem, there is a series of works going from the paper of Woeginger [8] to the paper of Bar-Noy et al. [4], who proposed on-line algorithms with constant competitive ratios. Note that the two degradable monocriterion intervals problem has been investigated in [2].

Outline of the Paper. In Section 1, we present two monocriterion algorithms (GOL^k [7] and LR^k [4]) in the degradable interval model. Section 2 is devoted to the description and the analysis of our on-line bicriteria algorithm AB^k using GOL and LR as subroutines.

1 Two On-Line Monocriteria Algorithms for the *size* and the *Proportional Weight* Metrics

In this section, we describe and analyze the competitiveness of the algorithm of Faigle and Nawijn [7] and the algorithm of Bar-Noy et al. [4]. We use them as subroutines of our algorithm AB^k (see Section 2) in order to obtain a pair of constant competitive ratios.

The Algorithm GOL^k. We describe the algorithm GOL^k of [7] in the degradable interval model by decomposing it into an interrupting stage and a scheduling stage.

Algorithm GOL^k (adaptation of [7])

When a new interval σ_i, defined by (r_i, q_i, d_i), is revealed, choose its effective execution $\sigma_i^q = [r_i, q_i)$ (i.e. each new interval is totally degraded) and do:

Interrupting stage: If there are k served intervals intersecting the date r_i, let σ_{max} be the one with the maximum deadline.
If σ_{max} does not exist (there is a free machine), do not interrupt any interval.
Else, If $d_{max} \geq q_i$ then interrupt σ_{max}.
If $d_{max} < q_i$ then do not interrupt any interval.

Scheduling stage:
If there is a free machine, then schedule σ_i^q on it.
Else, reject σ_i.

Note that the original algorithm of [7] is described for the classical non-degradable interval model. In the following, we denote by GOL_N^k this original version of the algorithm (notice that it is the same algorithm as GOL^k, except that GOL_N^k does not degrade any interval).

Lemma 1 *GOL^k is optimal for the Size in the degradable interval model.*

Proof. Let $\sigma_1, \ldots, \sigma_n$ be an on-line sequence of intervals such that for all i, $1 \leq i \leq n$, σ_i is defined by (r_i, q_i, d_i). Let $\sigma_1^D, \ldots, \sigma_n^D$ be the sequence such that for all i, $1 \leq i \leq n$, $\sigma_i^D = [r_i, q_i)$ (i.e. the sequence version with the intervals totally degraded). Let O_D^* be an optimal schedule of $\sigma_1, \ldots, \sigma_n$ in the degradable interval model and let O_N^* be an optimal schedule of $\sigma_1^D, \ldots, \sigma_n^D$ in the non-degradable interval model. By [2], we know that $N(O_D^*) = N(O_N^*)$. Furthermore, since GOL_N^k is optimal for the *Size* in the non-degradable interval model (See [7]), we have $N(O_N^*) = N(GOL_N^k(\sigma_1^D, \ldots, \sigma_n^D))$. By definition of GOL^k we have $GOL_N^k(\sigma_1^D, \ldots, \sigma_n^D) = GOL^k(\sigma_1, \ldots, \sigma_n)$. If we combine all these equalities, we obtain $GOL^k(\sigma_1, \ldots, \sigma_n) = GOL_N^k(\sigma_1^D, \ldots, \sigma_n^D) = N(O_N^*) = N(O_D^*)$. Thus, GOL^k is optimal for the *Size* in the degradable interval model. □

The Algorithm LR^k. We now describe the algorithm LR^k of [4] adapted to our degradable interval model and decomposed into an interrupting stage and a scheduling stage (for all $k \geq 3$).

Algorithm LR^k (adaptation of [4])

We denote by F_t the set of scheduled intervals containing date t. When a new interval σ_i defined by (r_i, q_i, d_i) is revealed, choose the effective execution $\sigma_i^d = [r_i, d_i)$ (i.e. do not degrade any interval) and do:

Interrupting stage:
If $|F_{r_i}| < k$, then do not interrupt any interval.
If $|F_{r_i}| = k$, then
1. Sort the $k+1$ intervals of $F_{r_i} \cup \{\sigma_i^d\}$ by increasing order of release dates. If several intervals have the same release date, order them in the decreasing order of their deadlines and let L be the set of the $\lceil \frac{k}{2} \rceil$ first intervals.
2. Sort the $k+1$ intervals of $F_{r_i} \cup \{\sigma_i^d\}$ by decreasing order of deadlines (ties are broken arbitrarily) and let R be the set of the $\lfloor \frac{k}{2} \rfloor$ first intervals.

If $\sigma_i^d \in L \cup R$, then interrupt any interval σ_j in $F_{r_i} - L \cup R$,
Else do not interrupt any interval.

Scheduling stage:
If $|F_{r_i}| < k$, then schedule σ_i^d on any free machine.
If $|F_{r_i}| = k$ and $\sigma_i^d \in L \cup R$, then schedule σ_i^d on the machine where σ_j has been interrupted.
If $|F_{r_i}| = k$ and $\sigma_i^d \notin L \cup R$, then reject σ_i.

In the following, we show that LR^k is $\left(\frac{4}{1-\frac{2}{k}}\right)$-competitive in the degradable interval model for the *Weight* metric (note that this is very close to 4 when k is large). We first show that the weight of an optimal degradable schedule is no more than twice the weight of an optimal non-degradable schedule.

Lemma 2 *For every set of intervals $\{\sigma_1, \ldots, \sigma_n\}$, let O^{*nd} be an optimal schedule of $\{\sigma_1, \ldots, \sigma_n\}$ for the proportional weight metric in the non-degradable interval model (i.e. $q_i = d_i$), and let O^{*d} be an optimal schedule of $\{\sigma_1, \ldots, \sigma_n\}$ for the same metric in the degradable interval model. We have:*

$$W(O^{*d}) \leq 2W(O^{*nd})$$

Proof. Let $O_1^{*d}, \cdots, O_k^{*d}$ be the k sub-schedules of O^{*d} (O_i^{*d} executes the same intervals at the same dates as machine i of O^{*d}). Thus, we have:

$$W(O^{*d}) = \sum_{i=0}^{k} W(O_i^*)$$

Let $\Gamma_i = \{\sigma_j \in \{\sigma_1, \ldots, \sigma_n\} : \sigma_j \in O_i^{*d}\}$. O_i^{*d} is an optimal schedule of Γ_i in the degradable interval model. Indeed, suppose, by contradiction, that there exists a valid schedule O of Γ_i such that $W(O_i^{*d}) < W(O)$. This means that the valid schedule consisting in the union of the O_j^{*d}'s, except for $j = i$, which is replaced by O, generates a weight greater than O^{*d} and is valid. This contradicts the optimality of O^{*d}.

Let us apply the 2-approximation algorithm for one machine schedules described in [2] separately on each Γ_i ($1 \leq i \leq k$). Let $O_1, \cdots, O_k$ be the obtained schedules. Thus, by Theorem 5 of [2], for each i, we have $W(O_i^{*d}) \leq 2W(O_i)$. We sum the k inequalities and we obtain $W(O^{*d}) \leq 2\sum_{i=1}^{k} W(O_i)$. Moreover, since the 2-approximation algorithm of [2] does not degrade the intervals, the k machine schedule consisting in the union of the O_i's is valid for the non-degradable interval model. This means that $\sum_{i=1}^{k} W(O_i) \leq W(O^{*nd})$. Combining this last inequality with $W(O^{*d}) \leq 2\sum_{i=1}^{k} W(O_i)$ leads to:

$$W(O^{*d}) \leq 2W(O^{*nd})$$

□

Corollary 1 *LR^k is $\left(\frac{4}{1-\frac{2}{k}}\right)$-competitive for the degradable interval model on $k \geq 3$ machines.*

Proof. It is known that LR^k is $(\frac{2}{1-\frac{2}{k}})$-competitive for the non-degradable interval model (from an adaptation of the proof of [4]). Thus, by definition, we have $\frac{2}{1-\frac{2}{k}}W(LR^k(\sigma)) \geq W(O^{*nd})$. By Lemma 2, we have $2W(O^{*nd}) \geq W(O^{*d})$. Combining these two inequalities leads to $\frac{4}{1-\frac{2}{k}}W(LR^k(\sigma)) \geq W(O^{*d})$. This means that LR^k is $\left(\frac{4}{1-\frac{2}{k}}\right)$-competitive for the degradable model. □

2 Our Algorithm AB^k

Definition 2 (Cover relation). *Let σ be an interval defined by the triplet (r, q, d). Let $\sigma^1 = [r, t_1)$ and $\sigma^2 = [r, t_2)$ be two valid effective executions of σ (i.e. $q \leq t_1 \leq d$ and $q \leq t_2 \leq d$). We say that σ^1* covers *σ^2 if and only if $\sigma^2 \subseteq \sigma^1$ (i.e. if and only if $t_2 \leq t_1$).*

Definition 3 (Union$^+$ of sets of degraded intervals). *Let $\{\sigma_1, \cdots, \sigma_n\}$ be a set of degradable intervals. For all i, $1 \leq i \leq n$, σ_i is defined by (r_i, q_i, d_i). For all $\sigma_i \in \{\sigma_1, \cdots, \sigma_n\}$, let $\sigma_i^1 = [r_i, t_i^1)$ and $\sigma_i^2 = [r_i, t_i^2)$ be two valid effective executions of σ_i (i.e. $q_i \leq t_i^1 \leq d_i$ and $q_i \leq t_i^2 \leq d_i$). Let $E_1 \subseteq \{\sigma_i^1 : \sigma_i \in \{\sigma_1, \ldots, \sigma_n\}\}$ and $E_2 \subseteq \{\sigma_i^2 : \sigma_i \in \{\sigma_1, \ldots, \sigma_n\}\}$. We define $E_1 \uplus E_2$ the* union$^+$ *of E_1 and E_2 as follows:*

- *$\sigma_i^1 \in E_1 \uplus E_2$ if and only if ($\sigma_i^1 \in E_1$ and $\sigma_i^2 \notin E_2$) or ($\sigma_i^1 \in E_1$ and $\sigma_i^2 \in E_2$ and σ_i^1 covers σ_i^2).*
- *$\sigma_i^2 \in E_1 \uplus E_2$ if and only if ($\sigma_i^2 \in E_2$ and $\sigma_i^1 \notin E_1$) or ($\sigma_i^2 \in E_2$ and $\sigma_i^1 \in E_1$ and σ_i^2 covers σ_i^1).*

Note that the union$^+$ is commutative and it generalizes the usual definition of the union of two non-degradable intervals sets since in that case, $\sigma_i^1 = \sigma_i^2$. Thus, for all σ, E_1 and E_2, if $\sigma \in E_1 \uplus E_2$, then $\sigma \in E_1 \cup E_2$.

As, by definition, the two effective executions of a same interval σ defined by (r, q, d) must start at the same release date r, the one with the smallest execution time is covered by the other.

Note that, to be completely rigorous, we should not define an interval σ_i by (r_i, q_i, d_i), but by (r_i, q_i, d_i, i). Indeed, let us consider the following problematic example. Let $\sigma_i^1 = [r_i, t_i^1)$ be an effective execution of $\sigma_i = (r_i, q_i, d_i)$ and $\sigma_j^1 = [r_j, t_j^1)$ be an effective execution of $\sigma_j = (r_j, q_j, d_j)$, with $i \neq j$. If we consider the particular case where $r_i = r_j$ and $t_i^1 = t_j^1$ (our model allows such a situation), then we have $\sigma_i^1 = [r_i, t_i^1) = [r_j, t_j^1) = \sigma_j^1$. Of course, in this paper, we consider that the intervals are *distinct* (i.e. $\sigma_i^1 \neq \sigma_j^1$). That is why we should define an interval σ_i by (r_i, q_i, d_i, i) and an effective execution σ_i^1 by $([r_i, t_i^1), i)$. But, in order to simplify the notations, we write $\sigma_i = (r_i, q_i, d_i)$ instead of $\sigma_i = (r_i, q_i, d_i, i)$, and $\sigma_i^1 = [r_i, t_i^1)$ instead of $\sigma_i^1 = ([r_i, t_i^1), i)$.

The Algorithm AB^k. The main idea is the following. AB^k is running on k identical machines (called *real* machines because it is on these machines that the effective schedule is built). It uses as subroutines GOL and LR (described in Section 1). For the ease of notation, we use A for GOL and B for LR. For each new submitted interval σ_i, we simulate the execution of the algorithm A^r (resp. B^{k-r}) on r (resp. $k-r$) *virtual* machines, in order to control the size (resp. the weight) of the schedule. These two simulations (for the size and for the weight) are made on machines that we call *virtual*, because they are used only in order to determine the set (potentially empty) of intervals AB^k has to interrupt and whether σ_i has to be rejected or served by AB^k (and in this last case, to decide in which degraded version AB^k has to serve the new interval). Indeed, AB^k serves σ_i on a *real* machine if and only if A^r or B^{k-r} serves σ_i (note that if both A^r and B^{k-r} serve it, AB^k chooses the effective execution of σ_i that covers the other).

In order to determine the schedule given by an algorithm after the interrupting and the scheduling stages, we introduce the following notation.

Notation 2 (Schedule returned by an algorithm on step i) *For every on-line sequence $\sigma_1, \ldots, \sigma_n$, for every algorithm ALG and for every step of execution i ($1 \leq i \leq n$) of ALG, let $\mathcal{O}_{i_1}(ALG)$ (resp. $\mathcal{O}_{i_2}(ALG)$) be the schedule returned ALG after the* interrupting *(resp.* scheduling*) stage of step i.*

Notation 3 (Set of intervals scheduled by AB^k) *For every on-line sequence $\sigma_1, \ldots, \sigma_n$, for every step of execution i ($1 \leq i \leq n$) of the algorithm AB^k, let $\mathcal{R}_{i_1}(AB^k)$ (resp. $\mathcal{R}_{i_2}(AB^k)$) be the set of intervals scheduled and not interrupted after the* interrupting *(resp. the* scheduling*) stage of step i on the k machines associated to AB^k, called* real *machines.*

Notation 4 (Set of intervals scheduled by A^r and B^{k-r}) *For every on-line sequence $\sigma_1, \ldots, \sigma_n$, for every step of execution i ($1 \leq i \leq n$) of the algorithm A^r (resp. B^{k-r}), let $\mathcal{V}_{i_1}(A^r)$ (resp. $\mathcal{V}_{i_1}(B^{k-r})$) be the set of intervals scheduled and not interrupted after the* interrupting stage *of step i on the r (resp $k-r$) machines associated to A^r (resp. B^{k-r}). Let $\mathcal{V}_{i_2}(A^r)$ (resp. $\mathcal{V}_{i_2}(B^{k-r})$) be the set of intervals scheduled and not interrupted after the* scheduling stage *of step i on the r (resp. $k-r$) machines associated to A^r (resp. B^{k-r}). The r (resp $k-r$) machines associated to A^r (resp. B^{k-r}) are called* virtual *machines.*

We give now a formal description of the algorithm AB^k.

Input: An on-line sequence of intervals $\sigma_1, \ldots, \sigma_n$ and k identical machines.

Output: After each step i ($1 \leq i \leq n$), a valid schedule $\mathcal{O}_{i_2}(AB^k)$ of $\sigma_1, \ldots, \sigma_i$ on the k real machines.

Step 0: $\mathcal{V}_{0_2}(A^r) = \mathcal{V}_{0_2}(B^{k-r}) = \mathcal{R}_{0_2}(AB^k) = \emptyset$

Step i (date r_i):

1. The **interrupting stage** of AB^k:

(a) Execute the *interrupting stage* of A^r (resp. B^{k-r}) on the r (resp. $k-r$) virtual machines associated to A^r (resp. B^{k-r}) by submitting the new interval σ_i to A^r (resp. B^{k-r}). Note that the set of intervals scheduled and not interrupted by A^r (resp. B^{k-r}) is now $\mathcal{V}_{i_1}(A^r)$ (resp. $\mathcal{V}_{i_1}(B^{k-r})$).

(b) Execute the *interrupting stage* of AB^k on the k real machines associated to AB^k by interrupting the subset of intervals of $\mathcal{R}_{(i-1)_2}(AB^k)$ such that: $\mathcal{R}_{i_1}(AB^k) = \mathcal{V}_{i_1}(A^r) \uplus \mathcal{V}_{i_1}(B^{k-r})$

2. The **scheduling stage** of AB^k:

(a) Execute the *scheduling stage* of A^r (resp. B^{k-r}) on the r (resp. $k-r$) virtual machines associated to A^r (resp. B^{k-r}) by serving or rejecting the new interval σ_i.

(b) Execute the *scheduling stage* of AB^k on the k real machines associated to AB^k by switching to the appropriate case:

i. If A^r and B^{k-r} reject σ_i, then AB^k does not schedule σ_i. Thus, we have: $\mathcal{R}_{i_2}(AB^k) = \mathcal{R}_{i_1}(AB^k)$

ii. If A^r serves σ_i (with effective execution σ_i^A) and B^{k-r} rejects σ_i then AB^k serves σ_i^A on any free machine and we have:
$$\mathcal{R}_{i_2}(AB^k) = \mathcal{R}_{i_1}(AB^k) \cup \{\sigma_i^A\}$$

iii. If A^r rejects σ_i and B^{k-r} serves σ_i (with effective execution σ_i^B) then AB^k serves σ_i^B on any free machine and we have:
$$\mathcal{R}_{i_2}(AB^k) = \mathcal{R}_{i_1}(AB^k) \cup \{\sigma_i^B\}$$

iv. If A^r and B^{k-r} serve σ_i one with effective execution σ_i^A and the other with effective execution σ_i^B then AB^k serves the effective execution that covers the other on any free machine. If σ_i^B covers σ_i^A then we have:
$$\mathcal{R}_{i_2}(AB^k) = \mathcal{R}_{i_1}(AB^k) \cup \{\sigma_i^B\}$$
else we have:
$$\mathcal{R}_{i_2}(AB^k) = \mathcal{R}_{i_1}(AB^k) \cup \{\sigma_i^A\}$$

Intervals Scheduled by the Algorithm AB^k. We first present Lemma 3 which states that the algorithm AB^k schedules the same intervals as the union$^+$ of the intervals scheduled by A^r and the intervals scheduled by B^{k-r}.

Lemma 3 *For each step i of execution of the algorithm AB^k, the schedule $\mathcal{O}_{i_2}(AB^k)$ is valid and $\mathcal{R}_{i_2}(AB^k) = \mathcal{V}_{i_2}(A^r) \uplus \mathcal{V}_{i_2}(B^{k-r})$.*

Proof. We prove Lemma 3 by induction on the steps of execution i of AB^k.
The basic case (step 0): By definition of AB^k, we have $\mathcal{V}_{0_2}(A^r) = \mathcal{V}_{0_2}(B^{k-r}) = \mathcal{R}_{0_2}(AB^k) = \emptyset$. Thus, $\mathcal{O}_{i_2}(AB^k)$ is valid and we have $\mathcal{R}_{i_2}(AB^k) = \mathcal{V}_{i_2}(A^r) \uplus \mathcal{V}_{i_2}(B^{k-r})$. The basic case is checked.
The main case (step i): Let us assume that $\mathcal{O}_{(i-1)_2}(AB^k)$ is valid and that $\mathcal{R}_{(i-1)_2}(AB^k) = \mathcal{V}_{(i-1)_2}(A^r) \uplus \mathcal{V}_{(i-1)_2}(B^{k-r})$ (by the assumption of the induction).

1. The interrupting stage: We first need to prove that $\mathcal{R}_{i_1}(AB^k) = \mathcal{V}_{i_1}(A^r) \uplus \mathcal{V}_{i_1}(B^{k-r})$ and that $\mathcal{O}_{i_1}(AB^k)$ is valid.

(a) By definition, AB^k interrupts the subset of intervals of $\mathcal{R}_{(i-1)_2}(AB^k)$ such that:

$$\mathcal{R}_{i_1}(AB^k) = \mathcal{V}_{i_1}(A^r) \uplus \mathcal{V}_{i_1}(B^{k-r}) \qquad \text{(UNION)}$$

We have to show that there is always a subset of intervals of $\mathcal{R}_{(i-1)_2}(AB^k)$ that can be removed such that the above equality is possible.

Since $\mathcal{V}_{i_1}(A^r) \subseteq \mathcal{V}_{(i-1)_2}(A^r)$, $\mathcal{V}_{i_1}(B^{k-r}) \subseteq \mathcal{V}_{(i-1)_2}(B^{k-r})$, and $\mathcal{R}_{(i-1)_2}(AB^k) = \mathcal{V}_{(i-1)_2}(A^r) \uplus \mathcal{V}_{(i-1)_2}(B^{k-r})$ (by the assumption of the induction), we have $\mathcal{V}_{i_1}(A^r) \uplus \mathcal{V}_{i_1}(B^{k-r}) \subseteq \mathcal{R}_{(i-1)_2}(AB^k)$.

(b) By definition, AB^k interrupts only intervals scheduled in $\mathcal{O}_{(i-1)_2}(AB^k)$, and by assumption of induction, $\mathcal{O}_{(i-1)_2}(AB^k)$ is valid. Thus, there cannot be intervals scheduled at the same time or more than once in $\mathcal{O}_{i_1}(AB^k)$. This means that $\mathcal{O}_{i_1}(AB^k)$ is valid. (VALID)

2. The scheduling stage: We now prove that $\mathcal{R}_{i_2}(AB^k) = \mathcal{V}_{i_2}(A^r) \uplus \mathcal{V}_{i_2}(B^{k-r})$ and that $\mathcal{O}_{i_2}(AB^k)$ is valid. By definition of AB^k, several cases may happen:

(a) If A^r and B^{k-r} reject σ_i, then AB^k does not schedule σ_i and we have:

i.
$$\begin{aligned}\mathcal{R}_{i_2}(AB^k) &= \mathcal{R}_{i_1}(AB^k) \;=\; \mathcal{V}_{i_1}(A^r) \uplus \mathcal{V}_{i_1}(B^{k-r}) \\ &\quad \text{(by the definition of } AB^k \text{ and by (UNION))} \\ &= \mathcal{V}_{i_2}(A^r) \uplus \mathcal{V}_{i_2}(B^{k-r}) \\ &\quad \text{(since } A^r \text{ and } B^{k-r} \text{ reject } \sigma_i\text{, we have} \\ &\quad \mathcal{V}_{i_1}(A^r) = \mathcal{V}_{i_2}(A^r) \text{ and } \mathcal{V}_{i_1}(B^{k-r}) = \mathcal{V}_{i_2}(B^{k-r}))\end{aligned}$$

ii. $\mathcal{O}_{i_2}(AB^k) = \mathcal{O}_{i_1}(AB^k)$. Thus $\mathcal{O}_{i_2}(AB^k)$ is valid (because by (VALID), $\mathcal{O}_{i_1}(AB^k)$ is valid).

(b) If A^r serves σ_i (with effective execution σ_i^A) and B^{k-r} rejects σ_i, then AB^k schedules σ_i^A on any free real machine at time r_i and we have:

i.
$$\begin{aligned}\mathcal{R}_{i_2}(AB^k) &= \mathcal{R}_{i_1}(AB^k) \cup \{\sigma_i^A\} \;=\; (\mathcal{V}_{i_1}(A^r) \uplus \mathcal{V}_{i_1}(B^{k-r})) \cup \{\sigma_i^A\} \\ &\quad \text{(by the definition of } AB^k \text{ and by (UNION))} \\ &= \mathcal{V}_{i_2}(A^r) \uplus \mathcal{V}_{i_2}(B^{k-r}) \\ &\quad \text{(union and union}^+ \text{ commute and since } A^r \text{ serves } \sigma_i \\ &\quad \text{and } B^{k-r} \text{rejects } \sigma_i\text{, we have } \mathcal{V}_{i_2}(A^r) = \mathcal{V}_{i_1}(A^r) \cup \{\sigma_i^A\} \\ &\quad \text{and } \mathcal{V}_{i_2}(B^{k-r}) = \mathcal{V}_{i_1}(B^{k-r}))\end{aligned}$$

ii. Since $\mathcal{O}_{i_1}(AB^k)$ is a valid schedule (by (VALID)) and $\mathcal{O}_{i_2}(AB^k)$ is built by AB^k by adding σ_i to $\mathcal{O}_{i_1}(AB^k)$ only once, the only reason for which $\mathcal{O}_{i_2}(AB^k)$ could not be valid would be because σ_i is scheduled by AB^k at time r_i whereas there is no free machine at time r_i, i.e. because there are at least $k+1$ intervals of $\mathcal{R}_{i_2}(AB^k)$ scheduled at time r_i by AB^k. Let us prove that this is impossible. Indeed, since A^r and B^{k-r} build at each time valid schedules, there are at most $r + k - r = k$ intervals of $\mathcal{V}_{i_2}(A^r) \uplus \mathcal{V}_{i_2}(B^{k-r})$ scheduled at time r_i by A^r and B^{k-r}, and thus, there are at most k intervals of $\mathcal{R}_{i_2}(AB^k)$ scheduled at time r_i by AB^k (because we just proved above that $\mathcal{R}_{i_2}(AB^k) = \mathcal{V}_{i_2}(A^r) \uplus \mathcal{V}_{i_2}(B^{k-r})$). Thus, $\mathcal{O}_{i_2}(AB^k)$ is a valid schedule.

(c) If A^r rejects σ_i and B^{k-r} serves σ_i (with effective execution σ_i^B), then AB^k schedules σ_i^B on any free real machine at time r_i.
   i. We prove that $\mathcal{R}_{i_2}(AB^k) = \mathcal{V}_{i_2}(A^r) \uplus \mathcal{V}_{i_2}(B^{k-r})$ in the same way that we prove it in 2(b)i, except that we replace σ_i^A by σ_i^B.
   ii. We prove that $\mathcal{O}_{i_2}(AB^k)$ is valid in the same way as in 2(b)ii.

(d) If A^r and B^{k-r} serve σ_i one with effective execution σ_i^A and the other with effective execution σ_i^B. Let σ_i^S (resp. σ_i^L) be the shortest (resp. longest) effective execution of σ_i. Without loss of generality, we suppose that A^r schedules σ_i^S and B^{k-r} schedules σ_i^L. By definition of the algorithm, AB^k schedules σ_i^L, and we have:
   i.

$$\begin{aligned}
\mathcal{R}_{i_2}(AB^k) &= \mathcal{R}_{i_1}(AB^k) \cup \{\sigma_i^L\} \;=\; (\mathcal{V}_{i_1}(A^r) \uplus \mathcal{V}_{i_1}(B^{k-r})) \cup \{\sigma_i^L\} \\
&\quad \text{(by definition of } AB^k \text{ and by (UNION))} \\
&= (\mathcal{V}_{i_1}(A^r) \uplus \mathcal{V}_{i_1}(B^{k-r})) \cup (\{\sigma_i^L\} \uplus \{\sigma_i^S\}) \\
&\quad \text{(because, by definition of union}^+\text{, } \{\sigma_i^L\} = \{\sigma_i^L\} \uplus \{\sigma_i^S\}) \\
&= \mathcal{V}_{i_1}(A^r) \uplus \mathcal{V}_{i_1}(B^{k-r}) \uplus \{\sigma_i^L\} \uplus \{\sigma_i^S\} \\
&\quad \text{(because } (\mathcal{V}_{i_1}(A^r) \uplus \mathcal{V}_{i_1}(B^{k-r})) \cap (\{\sigma_i^L\} \uplus \{\sigma_i^S\}) = \emptyset) \\
&= (\mathcal{V}_{i_1}(A^r) \cup \{\sigma_i^S\}) \uplus (\mathcal{V}_{i_1}(B^{k-r}) \cup \{\sigma_i^L\}) \\
&\quad \text{(because the union}^+ \text{ is commutative and since } \sigma_i^S \notin \\
&\quad \mathcal{V}_{i_1}(A^r), \text{ we have } \mathcal{V}_{i_1}(A^r) \uplus \{\sigma_i^S\} = \mathcal{V}_{i_1}(A^r) \cup \{\sigma_i^S\}) \\
&= \mathcal{V}_{i_2}(A^r) \uplus \mathcal{V}_{i_2}(B^{k-r}) \\
&\quad \text{(because since } A^r \text{ and } B^{k-r} \text{ serve } \sigma_i, \text{ we have } \mathcal{V}_{i_2}(A^r) \\
&\quad = \mathcal{V}_{i_1}(A^r) \cup \{\sigma_i^S\} \text{ and } \mathcal{V}_{i_2}(B^{k-r}) = \mathcal{V}_{i_1}(B^{k-r}) \cup \{\sigma_i^L\})
\end{aligned}$$

   ii. We prove that $\mathcal{O}_{i_2}(AB^k)$ is valid in the same way as in 2(b)ii.

□

Corollary 2 *Let $A^r = GOL^r$ and $B^{k-r} = LR^{k-r}$. Let $N(\mathcal{V}_{i_2}(GOL^r)) = |\mathcal{V}_{i_2}(GOL^r)|$ and $W(\mathcal{V}_{i_2}(LR^{k-r}))$ be the sum of the weight of the intervals of $\mathcal{V}_{i_2}(LR^{k-r})$. For every input sequence $\sigma_1, \ldots, \sigma_n$ and for every step i ($1 \leq i \leq n$) of the algorithm AB^k, we have:*

$$N(\mathcal{V}_{i_2}(GOL^r)) \leq N(\mathcal{R}_{i_2}(AB^k)) \quad \text{and} \quad W(\mathcal{V}_{i_2}(LR^{k-r})) \leq W(\mathcal{R}_{i_2}(AB^k))$$

Proof. By Lemma 3, for every step i of the algorithm AB^k, we have $\mathcal{R}_{i_2}(AB^k) = \mathcal{V}_{i_2}(A^r) \uplus \mathcal{V}_{i_2}(B^{k-r}) = \mathcal{V}_{i_2}(GOL^r) \uplus \mathcal{V}_{i_2}(LR^{k-r})$, thus, by definition of union$^+$, Corollary 2 is checked. □

Theorem 1. *For all $k \geq 4$, for all r, $1 \leq r \leq k-3$, the algorithm AB^k applied with GOL^r and LR^{k-r} is $\left(\frac{k}{r}, \frac{4k}{k-r-2}\right)$-competitive for the Size and Proportional weights metrics.*

Proof. Let $\sigma_1, \cdots, \sigma_n$ be any on-line sequence of intervals and let O_x^{N*} (resp. O_x^{W*}) be an optimal schedule of $\{\sigma_1, \cdots, \sigma_n\}$ for the size N (resp. for the proportional weight W) on $x \leq k$ machines. Let O_r^{GOL} (resp. O_{k-r}^{LR}) be the schedule returned by GOL^r (resp. LR^{k-r}) on the on-line sequence $\sigma_1, \cdots, \sigma_n$ on $r \leq k-3$ (resp. $k-r \geq 3$) machines. Since, by Lemma 1, GOL^r is 1-competitive (resp. by Corollary 1, LR^{k-r} is $\left(\frac{4}{1-\frac{2}{k-r}}\right)$-competitive), we have:

$$N(O_r^{N*}) \leq N(O_r^{GOL}) \quad (\text{resp. } W(O_{k-r}^{W*}) \leq \left(\frac{4}{1-\frac{2}{k-r}}\right) W(O_{k-r}^{LR})) \tag{1}$$

Let O'^N (resp. O'^W) be the r (resp. $k-r$) machine sub-schedule of O_k^{N*} (resp. O_k^{W*}) executing all the intervals appearing on the r (resp. $k-r$) machines of O_k^{N*} (resp. O_k^{W*}) generating the largest size (resp. weight). Since O'^N (resp. O'^W) is a r (resp. $k-r$) machine schedule, we have $N(O'^N) \leq N(O_r^{N*})$ (resp. $W(O'^W) \leq W(O_{k-r}^{W*})$), otherwise, O_r^{N*} (resp. O_{k-r}^{W*}) would not be an optimal schedule for the *size* (resp. *weight*). Combined with (1), we obtain:

$$\begin{gathered} N(O'^N) \leq N(O_r^{N*}) \leq N(O_r^{GOL}) \\ (\text{resp. } W(O'^W) \leq W(O_{k-r}^{W*}) \leq \left(\tfrac{4}{1-\frac{2}{k-r}}\right) W(O_{k-r}^{LR})) \end{gathered} \tag{2}$$

Since O'^N (resp. O'^W) is the r machine sub-schedule of O_k^{N*} (resp. O_k^{W*}) generating the largest size (resp. weight), the average size (resp. weight) per machine in O'^N (resp. O'^W) is larger than the average size (resp. weight) per machine in O_k^{N*} (resp. O_k^{W*}). Thus, we have $\frac{N(O_k^{N*})}{k} \leq \frac{N(O'^N)}{r} \Rightarrow N(O_k^{N*}) \leq \frac{k}{r} N(O'^N)$ (resp. $\frac{W(O_k^{W*})}{k} \leq \frac{W(O'^W)}{k-r} \Rightarrow W(O_k^{W*}) \leq \frac{k}{k-r} W(O'^W)$). Combined with (2), we obtain:

$$N(O_k^{N*}) \leq \frac{k}{r} N(O_r^{GOL}) \quad (\text{resp. } W(O_k^{W*}) \leq \frac{4k}{k-r-2} W(O_{k-r}^{LR})) \tag{3}$$

As $N(O_r^{GOL}) = N(\mathcal{V}_{i_2}(GOL^r))$ (resp. $W(O_{k-r}^{LR}) = W(\mathcal{V}_{i_2}(LR^{k-r}))$), by applying Corollary 2 on (3), we obtain:

$$\begin{gathered} N(O_k^{N*}) \leq \tfrac{k}{r} N(\mathcal{V}_{i_2}(GOL^r)) \leq \tfrac{k}{r} N(\mathcal{R}_{i_2}(AB^k)) \\ (\text{resp. } W(O_k^{W*}) \leq \tfrac{k}{r} W(\mathcal{V}_{i_2}(LR^{k-r})) \leq \tfrac{4k}{k-r-2} W(\mathcal{R}_{i_2}(AB^k)) \text{) } \end{gathered}$$

This means that AB^k is $(\frac{k}{r}, \frac{4k}{k-r-2})$-competitive. □

Example. If $r = \frac{k}{2}$ (if $\frac{k}{2} \geq 3$ and k even), AB^k is $\left(2, \frac{8}{1-\frac{4}{k}}\right)$-competitive.

References

1. E. Arkin and B. Silverberg, *Scheduling jobs with fixed start and end times*, Discrete Applied Mathematics, 18 (1987), pp. 1–8.
2. F. Baille, E. Bampis, and C. Laforest, *Maximization of the size and the weight of schedules of degradable intervals*, in proceedings of COCOON'04, K.-Y. Chwa and I. Munro, eds., LNCS No. 3106, Springer-Verlag, 2004, pp. 219–228.
3. ———, *A note on bicriteria schedules with optimal approximation ratios*, Parallel Processing Letters, 14 (2004), pp. 315–323.
4. A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber, *Bandwidth allocation with preemption*, SIAM J. Comput., 28 (1999), pp. 1806–1828.
5. A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University press, 1998.
6. M. C. Carlisle and E. L. Lloyd, *On the k-coloring of intervals*, Discrete Applied Mathemetics, 59 (1995), pp. 225–235.
7. U. Faigle and M. Nawijn, *Note on scheduling intervals on-line*, Discrete Applied Mathematics, 58 (1995), pp. 13–17.
8. G. J. Woeginger, *On-line scheduling of jobs with fixed start and end times*, Theor. Comput. Sci., 130 (1994), pp. 5–16.

Off-Line Algorithms for Minimizing Total Flow Time in Broadcast Scheduling

Wun-Tat Chan[1,⋆], Francis Y.L. Chin[1,⋆⋆], Yong Zhang[1], Hong Zhu[2], Hong Shen[3], and Prudence W.H. Wong[4,⋆⋆⋆]

[1] Department of Computer Science, University of Hong Kong, Hong Kong
{wtchan,chin,yzhang}@cs.hku.hk
[2] Department of Computer Science and Engineering, Fudan University, China
hzhu@fudan.edu.cn
[3] Graduate School of Information Science
Japan Advanced Institute of Science and Technology, Japan
shen@jaist.ac.jp
[4] Department of Computer Science, University of Liverpool, UK
pwong@csc.liv.ac.uk

Abstract. We study the off-line broadcast scheduling problem to minimize total (or average) flow time. Assume the server has k pages and the requests arrive at n distinct times, we give the first algorithm to find the optimal schedule for the server with a single channel, in $O(k^3(n+k)^{k-1})$ time. For m-channel case, i.e., the server can broadcast m different pages at a time where $m < k$, we find the optimal schedule in $O(n^{k-m})$ time when k and m are constants. In the single channel case, we also give a simple linear-time approximation algorithm to minimize average flow time, which achieves an additive $(k-1)/2$-approximation.

1 Introduction

In an on-demand broadcasting system, the server receives requests for pages from clients over time, and answers these requests by broadcasting (sending) the pages via the broadcast channels. After the server broadcasts a page, all pending requests for that page are satisfied. The scheduler of the server is to arrange the order of the page broadcasts so as to minimize the total (or average) flow time of the requests. In this paper we assume that time is discrete, all the pages have unit length and the server has m broadcast channels, i.e., at most m different pages can be broadcast at each time slot. We formalize the problem as follows. Assume that the server contains k pages, namely $P_0, P_1, \ldots, P_{k-1}$, requested by clients at integral time only. Let r_{ti} denote the number of requests for P_i at time t. For a schedule, let b_{ti} be the earliest time at or after time t when P_i is broadcast. The *flow time* of a request for P_i is $b_{ti} - t + 1$ where

⋆ This research was supported in part by Hong Kong RGC Grant HKU-5172/03E.
⋆⋆ This research was supported in part by Hong Kong RGC Grant HKU-7142/03E.
⋆⋆⋆ This research was supported in part by Nuffield Foundation Grant NAL/01004/G.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 318–328, 2005.

t is the time when P_i is requested. Suppose that the last requests arrive at time n. The total flow time of the schedule, which is to be minimized, is equal to $\sum_{t=0}^{n}\sum_{i=0}^{k-1} r_{ti}(b_{ti}-t+1)$. Note that an optimal schedule that minimizes the total flow time is also an optimal schedule that minimizes the average flow time. In this paper we consider the off-line version of the problem, in which the server is aware of all the requests in advance.

Previous work of the problem considered that the number of broadcast channels $m=1$ and the number of pages k is a variable. This problem was shown to be NP-hard by Erlebach and Hall [5]. Recently, Bansal et al. [1] gave an additive $O(\sqrt{k})$-approximation algorithm in minimizing the average flow time, yet their algorithm requires to solve a time-consuming linear programming. Besides, most of the previous works considered the resource augmentation setting. In the setting, an m-speed algorithm refers to an algorithm of a server with m broadcast channels and an m-speed c-approximation algorithm is an m-speed algorithm which produces the schedules with total (or average) flow time at most c times that of the schedule produced by the optimal 1-speed algorithm. Kalyanasundaram et al. [9] gave an $\frac{1}{\epsilon}$-speed $\frac{1}{1-2\epsilon}$-approximation algorithm for any fixed $\epsilon \in (0, \frac{1}{3}]$. Gandhi et al. [7] gave an $\frac{1}{\epsilon}$-speed $\frac{1}{1-\epsilon}$-approximation algorithm for any fixed $\epsilon \in (0, \frac{1}{2}]$. To match the performance of the 1-speed optimal algorithm, Erlebach and Hall [5] gave a 6-speed algorithm, which was improved to 4-speed [7] and then to 3-speed by Gandhi et al. [8]. The on-line version of the problem has been studied by Edmonds and Pruhs [3, 4]. Bartal and Muthukrishnan [2] have considered the problem with another objective function which is to minimize the maximum flow time.

The main result of this paper is to give the first optimal algorithm for the broadcast scheduling problem to minimize total flow time when the number of pages k is fixed. Based on a dynamic programming technique and the concave property of the optimization function, our algorithm constructs the optimal schedule for the case when $m=1$ in $O(k^3(n+k)^{k-1})$ time where the last requests arrive at time n. When k is a constant, the time complexity is $O(n^{k-1})$. We generalize this result in the m-channel case where a server has $2 \le m < k$ broadcast channels. We show that in this case the optimal schedule can be found in $O((k^3+\binom{k-1}{k-m})(n+\frac{k}{m})^{k-m})$ time, or $O(n^{k-m})$ time when k and m are constants. Note that the problem with multiple channels seems to be easier than the problem with single channel and the multi-channel problem is not NP-hard if $k-m$ is a constant. In addition, we also give a simple approximation result for the case of $m=1$. Different from all previous works that need to solve the time-consuming linear programming, we give a simple linear-time algorithm to achieve a tight additive $\frac{k-1}{2}$-approximation on minimizing the average flow time, i.e., the average flow time of the schedules produced by our algorithm is at most that of any schedule plus $(k-1)/2$. Although our approximation algorithm seems inferior than the additive $O(\sqrt{k})$-approximation algorithm [1], our algorithm is much simpler and has comparable performance as k is usually small in practice.

The rest of the paper is organized as follows. Section 2 presents the optimal algorithms. A linear-time optimal algorithm for $m=1$ and $k=2$ is given in

Section 2.1. For general k, the optimal algorithms for the cases of $m = 1$ and general m are given in Sections 2.2 and 2.3, respectively. Section 3 presents the approximation algorithm for $m = 1$ and general k.

2 Optimal Algorithms

2.1 Broadcast Scheduling for Two Pages

Assume we have $m = 1$ broadcast channel, and $k = 2$ pages, and the last request arrives at time n. Let F denote the minimum total flow time in satisfying all requests. Given that P_i is broadcast at time t where $i = 0$ or 1, for $0 \le t \le n+1$, let $F_i(t)$ denote the minimum total flow time in satisfying all the requests made at or after time t, i.e., the r_{ab} requests for P_b at time a for $t \le a \le n$ and $b \in \{0, 1\}$. Note that by definition $F_i(n + 1) = 0$ for $i = 0$ or 1 because there is no request after time n. Otherwise, for example, $F_i(0)$ is the minimum total flow time to satisfy all requests given that P_i is broadcast at time 0. As only P_0 or P_1 can be broadcast at time 0, we can see that $F = \min\{F_0(0), F_1(0)\}$. In the following we define $F_i(t)$ recursively. For the base case where $t = n + 1$,

$$F_0(n + 1) = 0 \quad \text{and} \quad F_1(n + 1) = 0.$$

In general, we consider $0 \le t \le n$. For $F_0(t)$, the optimal schedule must have P_0 broadcast at each time $t, t + 1, \ldots, s - 1$ for some $s \ge t + 1$, and then P_1 broadcast at time s. Thus,

$$F_0(t) = \min_{t+1 \le s \le n+1} \{c_1(s, t) + F_1(s)\}$$

where $c_1(s, t) = \sum_{i=t}^{s-1} (r_{i0} + r_{i1}(s - i + 1))$ is the total flow time in satisfying those requests arrived within time t to time $s - 1$ inclusively. Similarly,

$$F_1(t) = \min_{t+1 \le s \le n+1} \{c_0(s, t) + F_0(s)\}$$

where $c_0(s, t) = \sum_{i=t}^{s-1} (r_{i0}(s - i + 1) + r_{i1})$.

With $O(n)$-time preprocessing (see Lemmas 8 and 9 in the Appendix), functions $c_0(s, t)$ and $c_1(s, t)$ can be computed in constant time for any given s and t. Hence, the brute-force method to find F by computing all $F_i(t)$ for $0 \le i \le 1$ and $0 \le t \le n$ takes $O(n^2)$ time. However, we show that it can be done in linear time by using the algorithm of Galil and Park [6]. We say that a function $\tau()$ is *concave* if it satisfies the *quadrangle inequality*, i.e., $\tau(a, c)+\tau(b, d) \le \tau(b, c)+\tau(a, d)$ for $a \le b \le c \le d$. Galil and Park have the following theorem.

Theorem 1 (Galil and Park [6]). *Given a concave function $\tau(i, j)$ for integer $0 \le i \le j \le n$ and $E(0)$, the recurrence $E(j) = \min_{0 \le i < j}\{D(i) + \tau(i, j)\}$ for $1 \le j \le n$ can be solved in $O(n)$ time, if $D(i)$ can be computed in constant time.*

We show that our recurrences can be transformed to that of Theorem 1, and thus our recurrences can also be solved in linear time. We give the details for

the case of $F_0()$ and the case of $F_1()$ can be done similarly. Let $E(j) = F_0(n - j + 1)$ for $0 \leq j \leq n + 1$. We have the base case $E(0) = F(n + 1) = 0$. Let $w(i, j) = c_1(n - i + 1, n - j + 1)$ for $0 \leq i < j \leq n + 1$. We have the recurrence $E(j) = \min_{0 \leq i < j}\{D(i) + w(i, j)\}$ for $1 \leq j \leq n + 1$, where $D(i) = F_1(n - i + 1)$. Given that the relevant values of $F_1()$ (resp. $F_0()$) are already known when $D(i)$ is needed, $D(i)$ can be obtained in constant time. Lemma 1 shows that function $w(i, j)$ satisfies the quadrangle inequality. Therefore, by Theorem 1, we can find the optimal schedule in linear time, as given in Theorem 2.

Lemma 1. *The function* $w(i, j) = c_1(n-i+1, n-j+1)$ *(resp.* $c_0(n-i+1, n-j+1)$*) for integer* $0 \leq i < j \leq n + 1$ *satisfies the quadrangle inequality, i.e.,* $w(a, c) + w(b, d) \leq w(b, c) + w(a, d)$ *for integer* $a \leq b \leq c \leq d$.

Proof. We consider the case that $w(i, j) = c_1(n - i + 1, n - j + 1)$ and the case of $w(i, j) = c_0(n - i + 1, n - j + 1)$ can be proved similarly. By definition, $w(i, j) = c_1(n - i + 1, n - j + 1) = \sum_{x=n-j+1}^{n-i}(r_{x0} + r_{x1}(n - i - x + 2))$. We can see that $\sum_{x=n-c+1}^{n-a} r_{x0} + \sum_{x=n-d+1}^{n-b} r_{x0} = \sum_{x=n-d+1}^{n-a} r_{x0} + \sum_{x=n-c+1}^{n-b} r_{x0}$ and $\sum_{x=n-c+1}^{n-a} r_{x1}(n-a-x+2)+\sum_{x=n-d+1}^{n-b} r_{x1}(n-b-x+2) = \sum_{x=n-c+1}^{n-a} r_{x1}(n-a-x+2) + \sum_{x=n-d+1}^{n-c} r_{x1}(n-b-x+2) + \sum_{x=n-c+1}^{n-b} r_{x1}(n-b-x+2) \leq \sum_{x=n-d+1}^{n-a} r_{x1}(n-a-x+2)+\sum_{x=n-c+1}^{n-b} r_{x1}(n-b-x+2)$. The last inequality is due to $n-a \geq n-b$. Therefore we have $w(a, c)+w(b, d) \leq w(b, c)+w(a, d)$. □

Theorem 2. *The minimum total flow time of the* 2*-page broadcast scheduling problem with requests arriving at integer time* 0 *to time* n *can be computed in* $O(n)$ *time.*

2.2 Broadcast Scheduling for k Pages

In this section we consider the problem with a single broadcast channel and k pages, for any fixed integer k. Assuming that the last requests arrive at time n, we formulate the problem as a dynamic programming problem which is a generalization of that in Section 2.1. Each sub-problem in the dynamic programming can be specified by a k-dimensional vector $v = (v_0, \ldots, v_{k-1})$ where $0 \leq v_i \leq n+k-1$ and $v_i \neq v_j$ if $i \neq j$. Let $v_{min} = \min_{0 \leq i \leq k-1}\{v_i\}$. The sub-problem corresponding to v is to find the minimum total flow time for satisfying all the requests between v_{min} and n, i.e., the r_{tj} requests for P_j at time t for $v_{min} \leq t \leq n$ and $0 \leq j \leq k - 1$, with v_i being P_i's earliest broadcasting time. For example, when $k = 2$, $F_0(t)$ defined in Section 2.1 refers to the minimum total flow time over all sub-problems corresponding to the vectors $v = (t, t')$ with $t' > t$. For general k, there are $O((n+k)^k)$ possible k-dimensional vectors as well as sub-problems, the time complexity will be at least $\Omega((n + k)^k)$. In the following, we shall modify the definition of the vectors corresponding to the sub-problems slightly so that better than $O((n + k)^k)$ time can be achieved.

The vector $v = (v_0, \ldots, v_{k-1})$ is similar to what is defined earlier except that one of the v_i's value is unspecified, which is represented as "$*$". Suppose that $v_\alpha = *$, it means that in the sub-problem corresponding to v, the earliest

broadcasting time of P_α is not fixed, yet it cannot be earlier than that of all other pages. Let $v_{min} = \min_{1 \le j \le k-1 \ \& \ v_j \neq *}\{v_j\}$. The sub-problem corresponding to v, say with $v_\alpha = *$, is to find the minimum total flow time for satisfying all the requests between v_{min} and n with v_i being P_i's earliest broadcasting time for $i \neq \alpha$. P_α's earliest broadcasting time can be any possible value between $v_{min}+1$ and $n+k-1$, i.e., some $\beta \in C_v$ with $C_v = \{t \mid t \neq v_j$ for all $v_j \neq *$ and $v_{min}+1 \le t \le n+k-1\}$.

Let $F(v)$ denote the minimum total flow time of the sub-problem corresponding to v. We define $F(v)$ recursively as follows. In the base case, we have $v_{min} = n+1$ which is the largest possible value for v_{min} as v needs to specify $k-1$ distinct values between v_{min} and $n+k-1$. By definition,

$$F(v) = 0 \quad \text{for all } v \text{ with } v_{min} = n+1$$

because there is no request after time n. In general, we consider $0 \le v_{min} \le n$ and assume that $v_\alpha = *$. Although v_α is unspecified, it can take the value $\beta \in C_v$ only, i.e., P_α can be broadcast the earliest at some time $\beta \in C_v$ only. We have to consider the $|C_v|$ different cases of assigning a value $\beta \in C_v$ to v_α. Therefore, $F(v)$ equals the minimum total flow time among the sub-problems corresponding to v with v_α assigned time β, for each $\beta \in C_v$. Similar to the 2-page case, for each of these sub-problems, if $v_x = v_{min}$, then the optimal schedule must have P_x broadcast at each time $v_{min}, v_{min}+1, \ldots, s-1$ where $s = \min\{\beta, \min_{v_j \neq v_{min} \ \& \ v_j \neq *}\{v_j\}\}$ is the earliest broadcasting time among the pages other than P_x. Note that there is no pending request for P_x immediately after time $s-1$. Thus, each of these sub-problems depends on one "smaller" sub-problem which is corresponding to another vector $u = (u_0, \ldots, u_{k-1})$ constructed from v as follows.

$$u_i = \begin{cases} * & \text{for } i \text{ where } v_i = v_{min}, \\ \beta & \text{for } i \text{ where } v_i = *, \\ v_i & \text{otherwise,} \end{cases}$$

for each $\beta \in C_v$. To compute $F(v)$, we consider the $|C_v|$ different "smaller" sub-problems, i.e.,

$$F(v) = \min_{\text{Sub-problems } u \text{ derived from } v} \{F(u) + c(u,v)\}$$

where $c(u,v) = \sum_{i=0}^{k-1} \sum_{t=v_{min}}^{u_{min}-1} f_{ti}(u,v)$ is the total flow time of the r_{ti} requests for P_i at time t for $v_{min} \le t \le u_{min}-1$ and $0 \le i \le k-1$ and $f_{ti}(u,v)$ is the total flow time of the particular r_{ti} requests for P_i at time t, i.e.,

$$f_{ti}(u,v) = \begin{cases} r_{ti} & \text{for } i \text{ where } v_i = v_{min}, \\ r_{ti}(u_i - t + 1) & \text{otherwise.} \end{cases}$$

We give an analysis on a brute-force implementation in solving the above recurrence of $F(v)$. Then we show a faster implementation by generalizing the approach used in Section 2.1. Lemma 2 implies that there are $k(n+k)!/(n+1)!$ different sub-problems corresponding to a k-dimensional vector. Similar to the

case of $k = 2$ in Section 2.1, with $O(kn)$-time preprocessing, $\sum_{t=v_{min}}^{u_{min}-1} f_{ti}(u, v)$ can be computed in constant time for any given i, v_{min} and $u_{min} - 1$ (see Lemmas 8 and 9 in the Appendix) and thus $c(u, v)$ can be computed in $O(k)$ time. Since the number of sub-problems corresponding to u, derived from a given v, is $O(n)$, a particular $F(v)$ can be computed in $O(kn)$ time. Therefore, the brute-force method in finding F by computing all $F(v)$ of the subproblems corresponding to v takes $O(k^2n(n+k)!/(n+1)!)$, or concisely, $O(k^2(n+k)^k)$ time.

Lemma 2. *There are $k(n + k)!/(n + 1)!$ different k-dimensional vectors $v = (v_0, \ldots, v_{k-1})$ with $0 \le v_j \le n - k + 1$ for all $0 \le j \le k - 1$ except one v_j's value equals $*$ and $v_i \ne v_j$ for all $i \ne j$.*

Proof. Since each $v_j \ne *$ should have a distinct value between $v_{min} + 1$ and $n+k-1$, there are at most $\binom{n+k}{k-1}$ ways of choosing $k-1$ distinct values between 0 and $n + k - 1$. As these $k - 1$ distinct values have $k!$ ways of assigning to the k positions, there are $k(n + k)!/(n + 1)!$ different valid vectors of v. □

Note that up to this stage, there is no gain in the time complexity and $\Omega((n+k)^k)$ time is still needed. In order to improve the time complexity, concave property of the function $c(u, v)$ should be exploited as in Section 2.1. In the faster implementation, we group the sub-problems into k^2 groups. Two sub-problems corresponding to v and u, respectively, belong to the same group G_{xy}, for $0 \le x, y \le k - 1$, if $v_{min} = v_x$ and $u_{min} = u_x$, and $v_y = u_y = *$, i.e., the two sub-problems have the earliest broadcasting time for the same page P_x and the unspecified broadcasting time same for another page P_y. We further divide the sub-problems in each group into sub-groups. Two sub-problems corresponding to v and v', respectively, of the same group G_{xy} belong to the same sub-group if except P_x and P_y all other pages among the two sub-problems have the same corresponding earliest broadcasting time, i.e., $v_j = v'_j$ for all j with $j \ne x$ and $j \ne y$. There are $O(n)$ sub-problems in each sub-group and there are $(n + k - 1)!/(n + 1)!$ sub-groups in each group, as shown in the following lemma.

Lemma 3. *There are $(n + k - 1)!/(n + 1)!$ sub-groups in each group.*

Proof. WLOG, consider a particular group G_{01}. The number of sub-groups in G_{01} is equal to the number of ways in choosing $k - 2$ distinct values between 1 and $n+k-1$ for $v_2, v_3, \ldots, v_{k-1}$, i.e., $\binom{n+k-1}{k-2}$ ways. As these $k-2$ distinct values have $(k-2)!$ ways of assigning to the $k-2$ positions, there are $(n+k-1)!/(n+1)!$ sub-groups in G_{01}. Similarly, it applies to each of the other groups. □

We can compute $F(v)$ for all vectors v corresponding to the sub-problems of the same sub-group in $O(k(n + k))$ time. It is done by transforming the recurrence in this section to that of Theorem 1 in Section 2.1. WLOG, we consider a sub-group in G_{01}. For all vectors v corresponding to the sub-problems in the sub-group, we have $v_0 = v_{min}$ and $v_1 = *$ and all other v_j fixed. For ease of explanation, we assume that $v_j \in [n + 2, n + k - 1]$ for all $2 \le j \le k - 1$. (The assumption is not necessary in proving the correctness of our algorithm.) For $0 \le t \le n + 1$, let $E(t) = F(v)$ where $v_{min} = n - t + 1$. Then we have

$E(0) = F(v) = 0$ because $v_{min} = n + 1$. For $0 \le s \le n + 1$, let $D(s) = F(u)$ where $u_1 = n - s + 1$ and u is derived from v. For $0 \le s < t \le n + 1$, let $w(s, t) = c(u, v)$, which is given as follows.

$$w(s,t) = \sum_{i=n-t+1}^{n-s} r_{i0} + \sum_{i=n-t+1}^{n-s} r_{i1}(n-s-i+2) + \sum_{j=2}^{k-1} \sum_{i=n-t+1}^{n-s} r_{ij}(v_j - i + 1)$$

It can be verified that the function $w(s, t)$ satisfies the quadrangle inequality as in Lemma 4. Hence, by Theorem 1, all $F(v)$ for sub-problems corresponding to v of the same sub-group can be computed in $O(kn)$ time and all $F(v)$ for all sub-problems of "all" sub-groups can be computed in $O(kn \cdot k^2 \cdot (n+k-1)!/(n+1)!)$, or concisely, $O(k^3(n + k)^{k-1})$. Thus, we have Theorem 3.

Lemma 4. *The function $w(s, t)$ for $0 \le s < t \le n + 1$ satisfies the quadrangle inequality, i.e., $w(a, c) + w(b, d) \le w(b, c) + w(a, d)$ for $a \le b \le c \le d$.*

Proof. The proof is similar to that of Lemma 1, which will be given in the full paper. □

Theorem 3. *The minimum total flow time of the k-page broadcast scheduling problem with requests arriving at integral times 0 to n can be computed in $O(k^3(n + k)^{k-1})$ time.*

2.3 Broadcast Scheduling with Multiple Channels

Assume that there are m broadcast channels available to the server. At each time slot, a server can broadcast at most m different pages among the k pages, where $m < k$. WLOG we can assume that there is an optimal schedule that broadcasts exactly m different pages at each time slot.

We apply the framework in solving the dynamic programming problem in Section 2.2 to this problem. Each sub-problem in the dynamic programming can be specified by a k-dimensional vector $v = (v_0, \ldots, v_{k-1})$ where $0 \le v_i \le n + \lceil (k - m)/m \rceil$ because after time n we need at most $\lceil (k - m)/m \rceil$ time units to satisfy the pending requests. The sub-problem corresponding to v gives the minimum total flow time for satisfying all the requests between v_{min} and n, i.e., the r_{ij} requests for P_j at time i for $v_{min} \le i \le n$ and $0 \le j \le k - 1$ with v_j being P_j's earliest broadcasting time. Since we assume that an optimal schedule has m page broadcasts at each time slot, in particular the first time slot v_{min}, it is sufficient to consider only those vectors v corresponding to the sub-problems with m earliest page broadcasts at time v_{min}, i.e., there are exactly m v_j's values equal to v_{min}. Same as that in Section 2.2, we consider that every vector v has one of the v_j equal to $*$. For a vector v with $v_\alpha = *$, the possible values between $v_{min} + 1$ and $n + \lceil (k - m)/m \rceil$ that can be assigned to v_α are in $C_v = \{i \mid$ there are less than m v_j's values equal to $i\}$.

To compute $F(v)$ of a vector v with $v_\alpha = *$, we find the minimum total flow time among the sub-problems corresponding to v with v_α assigned value β for

each $\beta \in C_v$. For each of these sub-problems, if $v_{x_1} = v_{x_2} = \ldots = v_{x_m} = v_{min}$ for some $0 \leq x_1, \ldots, x_m \leq k-1$, then the optimal schedule must have all pages $P_{x_1}, \ldots, P_{x_m}$ broadcast at each of times $v_{min}, v_{min}+1, \ldots, s-1$ where $s = \min\{\beta, \min_{v_j \neq v_{min} \ \& \ v_j \neq *}\{v_j\}\}$ is the earliest broadcasting time among the pages other than $P_{x_1}, \ldots, P_{x_m}$. Note that there is no pending request for $P_{x_1}, \ldots, P_{x_m}$ immediately after time $s-1$. Thus, each of these sub-problems depends on one "smaller" sub-problem which corresponds to a *relaxed* k-dimensional vector $\tilde{u}$ in which exactly m $\tilde{u}_j$'s values equal to $*$. We construct $\tilde{u}$ as follows.

$$\tilde{u}_i = \begin{cases} * & \text{for } i \text{ where } v_i = v_{min}, \\ \beta & \text{for } i \text{ where } v_i = *, \\ v_i & \text{otherwise.} \end{cases}$$

Let $\tilde{u}_{min} = \min_{\tilde{u}_j \neq *}\{\tilde{u}_j\}$. The sub-problem corresponding to $\tilde{u}$ is to find the minimum total flow time, denoted as $F(\tilde{u})$, to satisfy all requests between $\tilde{u}_{min}$ and n, i.e., the r_{ij} requests for P_j at time i for $u_{min} \leq i \leq n$ and $0 \leq j \leq k-1$, with $\tilde{u}_j$ being P_j's earliest broadcasting time for $\tilde{u}_j \neq *$. We do not need to compute $F(\tilde{u})$ directly. In fact $F(\tilde{u}) = \min\{F(v) \mid \tilde{u}_{min} = v_{min}$ and $\tilde{u}_j = v_j$ for all $\tilde{u}_j \neq *\}$, which is one of $F(v')$ for which the two sub-problems, corresponding to $\tilde{u}$ and v', respectively, have all pages having the same corresponding earliest broadcasting times except those pages P_j with $\tilde{u}_j = *$. We define the recurrence of $F(v)$ as follows.

$$F(v) = \min_{\text{sub-problems } \tilde{u} \text{ derived from } v} \{F(\tilde{u}) + c(\tilde{u}, v)\}$$

where $c(\tilde{u}, v) = \sum_{i=0}^{k} \sum_{t=v_{min}}^{\tilde{u}_{min}-1} f_{ti}(\tilde{u}, v)$ is the total flow time of the r_{ti} requests for P_i at time t for $0 \leq i \leq k-1$ and $v_{min} \leq t \leq \tilde{u}_{min} - 1$ and $f_{ti}(\tilde{u}, v)$ is the flow time of the particular r_{ti} requests for P_i at time t, i.e.,

$$f_{ti}(\tilde{u}, v) = \begin{cases} r_{ti} & \text{for } i \text{ where } v_i = v_{min}, \\ r_{ti}(\tilde{u}_i - t + 1) & \text{otherwise.} \end{cases}$$

We give an analysis on a brute-force implementation in solving the above recurrence of $F(v)$, and then we show a faster implementation. Lemma 5 implies that there are $O(k(n + k/m)^{k-m})$ different sub-problems we need to consider.

Lemma 5. *There are at most $O(k(n + k/m)^{k-m})$ different k-dimensional vectors $v = (v_0, \ldots, v_{k-1})$ satisfying the following conditions: (i) For all $0 \leq i \leq k-1$, $0 \leq v_j \leq n + k - 1$ except that one v_j's value is equal to $*$; (ii) for each $0 \leq t \leq n$, there are at most m v_j's values equal to t; and (iii) exactly m out of k v_j's values equal v_{min}.*

Proof (Sketch). Consider those vectors with that $v_0 = *$. As there are m v_j's values equal to v_{min}, the number of different vectors is bounded by the number of ways in choosing $k - m$ values, not necessarily distinct, between 0 and $n + \lceil (k-m)/m \rceil$, which is $O((n+k/m)^{k-m})$. Therefore the total number of different vectors is at most $O(k(n + k/m)^{k-m})$. □

If all $F(\tilde{u})$ are known and can be retrieved in constant time, then each computation of $F(v)$ takes $O(kn)$ time because computing $c(\tilde{u}, v)$ takes $O(k)$ time and there are $O(n)$ different $\tilde{u}$ to be considered. We can compute all $F(\tilde{u})$ as follows. Since $F(\tilde{u}) = \min\{F(v) \mid \tilde{u}_{min} = v_{min}$ and $\tilde{u}_j = v_j$ for all $\tilde{u}_j \neq *\}$, after each $F(v)$ is computed we update the corresponding values of $F(\tilde{u})$ where $\tilde{u}_{min} = v_{min}$ and $\tilde{u}_j = v_j$ for all $\tilde{u}_j \neq *$, if $F(v) < F(\tilde{u})$. It takes $O(\binom{k-1}{k-m})$ time to update each $F(v)$ because there are $\binom{k-1}{k-m}$ corresponding $\tilde{u}$, as shown in the Lemma 6. Therefore, it takes $O(kn + \binom{k-1}{k-m})$ time to compute each $F(v)$, hence $O((kn + \binom{k-1}{k-m})k(n+k)^{k-m})$ time to compute all values of $F(v)$ in the brute-force implementation.

Lemma 6. *For a k-dimensional vector v with exactly one v_j's value equals to $*$, there are $\binom{k-1}{k-m}$ k-dimensional (relaxed) vectors $\tilde{u}$ with exactly m $\tilde{u}_j$'s values equal to $*$ where $\tilde{u}_{min} = v_{min}$ and $\tilde{u}_j = v_j$ for all $\tilde{u}_j \neq *$.*

Proof (Sketch). It is equivalent to choosing $k - m$ of the $k - 1$ v_j's values with $v_j \neq *$. □

Similar to the efficient implementation in Section 2.2, we can partition the sub-problems $F(v)$ into groups and sub-groups so that we can apply the algorithm of Galil and Park [6] to compute all $F(v)$ of v corresponding to the sub-problems in a sub-group in $O(kn)$ time. Thus the overall time complexity for computing all $F(v)$ of v corresponding to the sub-problems of all sub-groups in all groups is $O((k^3 + \binom{k-1}{k-m})(n + k/m)^{k-m})$. When k and m are constants, the time complexity becomes $O(n^{k-m})$.

Theorem 4. *The minimum total flow time of the k-page broadcast scheduling problem with m broadcast channels where requests arriving at integral time 0 to time n can be computed in $O((k^3 + \binom{k-1}{k-m})(n + k/m)^{k-m})$ time, or $O(n^{k-m})$ time if k and m are constants.*

3 Approximation Algorithms

We present an approximation algorithm for the problem in minimizing the average flow time for the case when the number of broadcast channels $m = 1$. Note that the average flow time of a schedule equals the total flow time of the schedule divided by the total number of requests. Assuming that there are k pages that can be requested by clients, we give a simple algorithm with an additive approximation ratio of $(k - 1)/2$.

When there are only two pages, P_0 and P_1, for clients to request, i.e., $k = 2$. The algorithm considers two particular schedules, $S_0 = (P_0, P_1, P_0, P_1, \ldots)$ and $S_1 = (P_1, P_0, P_1, P_0, \ldots)$. We show in Lemma 7 that either S_0 or S_1 has the average flow time at most $1/2$ more than that of the optimal schedule. Thus, the algorithm achieves an additive approximation ratio of $1/2$ by choosing among S_0 and S_1 the schedule with a smaller total flow time.

Lemma 7. *Either S_1 or S_2 has the average flow time at most $1/2$ more than that of the optimal schedule.*

Proof. Let r_{i0} and r_{i1} be the number of requests for P_0 and P_1, respectively, at time i for $0 \le i \le n$. Let $R = \sum_{i=0}^{n}(r_{i0}+r_{i1})$ be the total number of requests. The average flow time of S_0 is $T(S_0) = 1+W_0/R$ where $W_0 = \sum_{i=1}^{n} r_{i,(i+1) \bmod 2}$ and the average flow time of S_1 is $T(S_1) = 1 + W_1/R$ where $W_1 = \sum_{i=0}^{n} r_{i,i \bmod 2}$. Since each page broadcast requires one flow time unit, the minimum average flow time $T^* \ge 1$. As $R = W_0 + W_1$, and thus $\min\{W_0/R, W_1/R\} \le 1/2$ and $\min\{T(S_0), T(S_1)\} \le T^* + 1/2$. □

For the problem of k pages $P_0, P_1, \ldots, P_{k-1}$, we consider the following k schedules which broadcast each page cyclically with different starting pages: $S_0 = (P_0, P_1, \ldots, P_{k-1}, P_0, P_1, \ldots, P_{k-1}, \ldots)$, $S_1 = (P_1, \ldots, P_{k-1}, P_0, P_1, \ldots, P_{k-1}, P_0, \ldots)$, $\ldots$ $S_{k-1} = (P_{k-1}, P_0, \ldots, P_{k-2}, P_{k-1}, P_0, \ldots, P_{k-2}, \ldots)$. Again the approximation algorithm is to choose the minimum average flow time schedule among these k schedules.

Theorem 5. *The minimum average flow time among schedules $S_0, \ldots, S_{k-1}$ is at most $(k-1)/2$ more than that of the optimal schedule*

Proof. Let $W_i = \sum_{i=0}^{n} r_{i,(i+j) \bmod k}$ for $0 \le j \le k-1$ and $R = \sum_{i=0}^{k-1} W_i$ be the total number of requests. The average flow time of S_j for $0 \le j \le k-1$ is $T(S_j) = 1 + \sum_{i=1}^{k-1}(i \cdot W_{(i+j) \bmod k})/R$, and the minimum among them is at most $\sum_{j=0}^{k-1} T(S_j)/k = 1 + \sum_{j=0}^{k-1}\sum_{i=0}^{k-1} W_{(i+j) \bmod k}/(kR) = 1 + k(k-1)\sum_{j=0}^{k-1} W_j/(2kR) \le T^* + (k-1)/2$, where $T^* \ge 1$ is the average flow time of the optimal algorithm. □

References

1. N. Bansal, S. K. M. Charikar, and J. Naor. Approximating the aversage response time in broadcast scheduling. In *SODA 2005*.
2. Y. Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *SODA 2000*, pages 558–559.
3. J. Edmonds and K. Pruhs. Multicast pull scheduling: When fairness is fine. *Algorithmica*, 36(3):315–330, 2003.
4. J. Edmonds and K. Pruhs. A maiden analysis of longest wait first. In *SODA 2004*, pages 818–827.
5. T. Erlebach and A. Hall. NP-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. In *SODA 2002*, pages 194–202.
6. Z. Galil and K. Park. A linear-time algorithm for concave one-dimensional dynamic programming. *Inf. Process. Lett.*, 33(6):309–311, 1990.
7. R. Gandhi, S. Khuller, Y. A. Kim, and Y.-C. J. Wan. Algorithms for minimizing response time in broadcast scheduling. *Algorithmica*, 38(4):597–608, 2004.
8. R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding in bipartite graphs. In *FOCS 2002*, pages 323–332.
9. B. Kalyanasundaram, K. R. Pruhs, and M. Velauthapillai. Scheduling broadcasts in wireless networks. *Journal of Scheduling*, 4(6):339–354, 2001.

Appendix

Lemma 8. *Given a sequence of $n+1$ numbers, $a_0, \ldots, a_n$, with $O(n)$ time preprocessing, we can compute $\sum_{k=i}^{j} a_k$ for any $0 \leq i \leq j \leq n$ in constant time.*

Proof. As all the prefix sums $b_i = \sum_{k=0}^{i} a_k$ can be computed in $O(n)$ time, each of the partial sums $\sum_{k=i}^{j} a_k = b_j - b_{i-1}$ can be computed in constant time. □

Lemma 9. *Given a sequence of $n+1$ numbers $a_0, \ldots, a_n$, with $O(n)$ time preprocessing, we can compute $\sum_{k=i}^{j} a_k(d-k+1)$ for any $0 \leq i \leq j \leq n$ and $j \leq d$ in constant time.*

Proof. As all the prefix sums $b_i = \sum_{k=0}^{i} a_k$ and weighted prefix sums $w_i = \sum_{k=0}^{i} a_k(n-i+1)$ can be computed in $O(n)$ time, each of the functions $\sum_{k=i}^{j} a_k(d-k+1) = w_j - w_{i-1} + (d-n)(b_j - b_{i-1})$, can be computed in constant time. □

Oblivious and Adaptive Strategies for the Majority and Plurality Problems

Fan Chung[1,⋆], Ron Graham[1,⋆⋆], Jia Mao[1,⋆⋆⋆], and Andrew Yao[2,†]

[1] Department of Computer Science and Engineering
University of California, San Diego
[2] Department of Computer Science, Tsinghua University, China

Abstract. In the well-studied *Majority problem*, we are given a set of n balls colored with two or more colors, and the goal is to use the minimum number of color comparisons to find a ball of the majority color (i.e., a color that occurs for more than $\lceil n/2 \rceil$ times). The *Plurality problem* has exactly the same setting while the goal is to find a ball of the dominant color (i.e., a color that occurs most often). Previous literature regarding this topic dealt mainly with adaptive strategies, whereas in this paper we focus more on the oblivious (i.e., non-adaptive) strategies. Given that our strategies are oblivious, we establish a linear upper bound for the *Majority problem* with arbitrarily many different colors. We then show that the *Plurality problem* is significantly more difficult by establishing quadratic lower and upper bounds. In the end, we also discuss some generalized upper bounds for adaptive strategies in the *k-color Plurality problem*.

1 Introduction

The *2-color Majority problem* was first raised by J. Moore in 1982 in connection with problems in the design of fault-tolerant computer systems. (It appeared in an equivalent setting of finding the majority vote among n processors with minimum number of paired comparisons[14]). In the colored-ball setting, we are given a set of n balls, each of which is colored in one of $k \in Z^+$ possible colors $\phi = \{c_1, c_2, ..., c_k\}$. We can choose any two balls a and b and ask questions of the form "Do a and b have the same color?". Our goal is to identify a ball of the *majority color* (i.e., meaning that this color occurs more than half of the time) or determine there is no majority color, using minimum possible number of questions.

We can view this problem as a game played between two players: **Q**, the Questioner, and **A**, the adversary. **Q**'s role is to ask a sequence of queries $Q(a, b)$:= "Is $\phi(a) = \phi(b)$?". **A** can answer each such query with the hope

⋆ Research supported in part by NSF Grant DMS 0100472 and ITR 0205061
⋆⋆ Research supported in part by NSF Grant CCR-0310991
⋆⋆⋆ Research supported in part by NSF Graduate Fellowship
† Research supported in part by CCR-0310466 and CCF-0426582

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 329–338, 2005.

of extending the game as long as possible before **Q** can finally identify a ball of the majority color or determine there is no majority. For the case when $k = 2$, a number of proofs were given (see Saks and Werman[15], Alonso, Reingold, Scott[5], and Wiener[17]) showing that $n - w_2(n)$ color comparisons are necessary and sufficient in the worst case, where $w_2(n)$ is the number of 1's in the binary representation of n.

Recently, several variants of this problem were also analyzed by Aigner[1]. One natural generalization is the so-called *Plurality problem* where the goal is just to identify a ball of the *dominant color* (i.e., meaning that this color occurs more often than any other color). This variant seems to be more difficult variant and only recently linear upper and lower bounds were given for $k = 3$ colors[2, 3].

In general, two types of strategies can be considered for **Q**. These are *adaptive* strategies in which each query can depend on the answers given to all previous queries, and *oblivious* (or non-adaptive) strategies in which all the queries must be specified before **A** is required to answer any of them. Clearly, in the oblivious case, **A** has more opportunity to be evasive. To the best of our knowledge, except for the case when $k = 2$[1], very little is known about the bounds for oblivious strategies for other variants of the *Majority problem.*

Let $MO_*(n)$ denote the minimum number of queries needed by **Q** in the *Majority problem* for arbitrarily many colors (i.e., k is not fixed) over all Oblivious strategies, $PO_k(n)$ (resp. $PA_k(n)$) the corresponding minimum in the *Plurality problem* for k colors over all oblivious (resp. adaptive) strategies. In this paper we will establish a linear upper bound for $MO_*(n)$ assuming a majority color exists, quadratic bounds for $PO_k(n)$, and also a generalized linear upper bound for $PA_k(n)$.

2 Oblivious Strategy for the Majority Problem

Consider the case where the number of possible colors k is unrestricted. In principle, this is a more challenging situation for **Q**. At least the upper bounds we have in this case are weaker than those for $k = 2$ colors[9]. A linear upper bound can be shown assuming the existence of a majority color. We also remark that without such assumption, a quadratic lower bound can be proven to be very close to the worst-case $\binom{n}{2}$ upper bound using similar argument as in the proof of Theorem 2 in Section 3. In fact, this quadratic lower bound can also be extended to the general case where the goal is to identify a ball whose color has appeared more than $t \geq \lceil n/2 \rceil$ times.

Theorem 1. *For all n,*

$$MO_*(n) \leq (1 + o(1))27n.$$

assuming a majority color exists.

Proof. Suppose the majority color is c. Let B denote the set of remaining colors $\{c_1, c_2, \ldots, c_R\}$ where R is unknown. Thus, v is not of majority color if and

only if $\phi(v) \in B$. We will specify the queries of $\mathbf{Q}$ by a graph H on the vertex set V, where an edge $\{v_i, v_j\}$ in H corresponds to the query $Q(v_i, v_j) :=$ "Is $\phi(v_i) = \phi(v_j)$?". The edge is colored *blue* if they are equal, and *red* if they are not equal. By a *valid assignment* ϕ on S we mean a mapping $\phi : S \to \{c, c_1, \ldots, c_R\}$ such that:
(i) $\phi(v_i) = \phi(v_j) \Rightarrow \{v_i, v_j\}$ is blue,
(ii) $\phi(v_i) \neq \phi(v_j) \Rightarrow \{v_i, v_j\}$ is red,
(iii) $|\phi^{-1}(c)| > n/2$.

We are going to use certain special graphs $X^{p,q}$, called Ramanujan graphs, which are defined for any primes p and q congruent to 1 modulo 4 (see [13]).

$X^{p,q}$ has the following properties:
(i) $X^{p,q}$ has $n = \frac{1}{2}q(q^2-1)$ vertices;
(ii) $X^{p,q}$ is regular of degree $p+1$;
(iii) The adjacency matrix of $X^{p,q}$ has the large eigenvalue $\lambda_0 = p+1$ and all other eigenvalues λ_i satisfying $|\lambda_i| \leq 2\sqrt{p}$.

We will use the following discrepancy inequality (see [4][8]) for a d-regular graph $H = H(n)$ with eigenvalues satisfying

$$\max_{i \neq 0} |\lambda_i| \leq \delta.$$

For any subset $X, Y \subseteq V(H)$, the vertex set of H, we have

$$| e(X,Y) - \frac{d}{n}|X|\ |Y|\ | \leq \frac{\delta}{n}\sqrt{|X|(n-|X|)|Y|(n-|Y|)} \tag{1}$$

where $e(X,Y)$ denotes the number of edges between X and Y.

Applying (1) to $X^{p,q}$, we obtain for all $X, Y \subseteq V(X^{p,q})$,

$$| e(X,Y) - \frac{p+1}{n}|X|\ |Y|\ | \leq \frac{2\sqrt{p}}{n}\sqrt{|X|(n-|X|)|Y|(n-|Y|)} \tag{2}$$

where $n = \frac{1}{2}q(q^2-1) = |V(X^{p,q})|$.

Now construct a Ramanujan graph $X^{p,q}$ on our vertex set $V = \{v_1, \ldots, v_n\}$. Let ϕ be a valid assignment of V to $\{c, c_1, \ldots, c_R\}$ and consider the subgraph G of $X^{p,q}$ induced by $\phi^{-1}(c)$ (the majority-color vertices of $X^{p,q}$ under the mapping ϕ).

Claim: Suppose $p \geq 41$. Then G has a connected component C with size at least $c'n$, where

$$c' > \frac{1}{2} - \frac{8p}{(p-1)^2}$$

Proof: We will use (2) with $X = C$, the largest connected component of G, and $Y = \phi^{-1}(c) \setminus X$. Write $|\phi^{-1}(c)| = \alpha n$ and $|C| = \beta n$. Since $e(X,Y) = 0$ for this choice, then by (2) we have

$$\begin{aligned}(p+1)^2|X|\ |Y| &\leq 4p(n-|X|)(n-|Y|),\\ (p+1)^2\beta(\alpha-\beta) &\leq 4p(1-\beta)(1-\alpha+\beta),\\ \beta(\alpha-\beta) &\leq \frac{4(1-\alpha)p}{(p-1)^2},\end{aligned}$$

There two possibilities:

$$\beta \geq \frac{1}{2}\left(\alpha + \sqrt{\alpha^2 - \frac{16(1-\alpha)p}{(p-1)^2}}\ \right) \text{ or } \beta \leq \frac{1}{2}\left(\alpha - \sqrt{\alpha^2 - \frac{16(1-\alpha)p}{(p-1)^2}}\ \right)$$

Subcase (a).

$$\begin{aligned}\beta &\geq \frac{1}{2}\left(\alpha + \sqrt{\alpha^2 - \frac{16(1-\alpha)p}{(p-1)^2}}\ \right)\\ &> \frac{1}{4}\left(1 + \sqrt{1 - \frac{32p}{(p-1)^2}}\ \right) \qquad \text{since } \alpha \geq 1/2\\ &\geq \frac{1}{2} - \frac{8p}{(p-1)^2} \qquad \text{since } p \geq 37\end{aligned}$$

as desired.
Subcase (b).

$$\begin{aligned}\beta &\leq \frac{1}{2}\left(\alpha - \sqrt{\alpha^2 - \frac{16(1-\alpha)p}{(p-1)^2}}\ \right)\\ &\leq \frac{8(1-\alpha)p}{\alpha(p-1)^2}\end{aligned}$$

Thus, we can choose a subset F of some of the connected components whose union $\cup F$ has size $xn = |\cup F|$ satisfying

$$\frac{\alpha}{2} - \frac{4(1-\alpha)p}{\alpha(p-1)^2} \leq x < \frac{\alpha}{2} + \frac{4(1-\alpha)p}{\alpha(p-1)^2} \tag{3}$$

Now we apply the discrepancy inequality (2) again by choosing $X = \cup F$ and $Y = \phi^{-1}(c) \setminus X$. We have

$$\begin{aligned}(p+1)^2 x(\alpha - x) &\leq 4p(1-x)(1-\alpha+x)\\ \text{or } x(\alpha - x) &\leq \frac{4(1-\alpha)p}{(p-1)^2}.\end{aligned}$$

However, it is easily checked that because of (3) this is not possible for $\alpha \geq 1/2$ and $p \geq 41$. Hence, subcase (b) cannot occur. This proves the claim.

Now we prove Theorem 1. Let $n = \frac{1}{2}q(q^2-1)$. Consider an algorithm $\mathbf{Q}$ specified by a graph $H = X^{p,q}$ where $p \geq 53$. We show that a good element can always be identified after all the queries are answered. Suppose we have an arbitrary blue/red coloring of the edges of $X^{p,q}$ with $p \geq 53$, and $\phi : S \to \{c, c_1, \ldots, c_R\}$ is a valid assignment on $V = V(X^{p,q})$. Consider the connected components formed by the blue edges of $X^{p,q}$. By the Claim there is at least one blue component of size at least $(\frac{1}{2} - \frac{8p}{(p-1)^2})n > \frac{1}{3}n$ (since $p \geq 53$). Call any such blue component *large*.

If there is only one large component then we are done, i.e., every point in it must be good. Since $p \geq 53$, there cannot be three large blue components. So the only remaining case is that we have exactly two large blue components, say S_1 and S_2. Again, if either $S_1 \subseteq \phi^{-1}(c)$ or $S_2 \subseteq \phi^{-1}(c)$ is forced, then we are done. So we can assume there is a valid assignment ϕ_1 with $S_1 \subseteq \phi_1^{-1}(c)$, $S_2 \subseteq \phi_1^{-1}(B)$, and a valid assignment ϕ_2 with $S_2 \subseteq \phi_2^{-1}(c), S_1 \subseteq \phi_2^{-1}(B)$ (where we recall that $B = \{c_1, c_2, \ldots, c_R\}$).

Let us write $S_i' = \phi_i^{-1}(c) \setminus S_i, i = 1, 2$. Clearly we must have $A := S_1' \cap S_2' \neq \emptyset$. Also note that $|A| \leq n - |S_1| - |S_2| < \frac{16p}{(p-1)^2} n$.

Define $B_1 = S_1' \setminus A, B_2 = S_2' \setminus A$. Observe that there can be no edge between A and $S_1 \cup S_2 \cup B_1 \cup B_2$. Now we are going to use (2) again, this time choosing $X = A, Y = S_1 \cup S_2 \cup B_1 \cup B_2$. Note that

$$n > |Y| = |\phi_1^{-1}(c)| - |A| + |\phi_2^{-1}(c)| - |A| > n - 2|A|.$$

Since $e(X, Y) = 0$, we have by (2),

$$\begin{aligned} (p+1)^2 |X|\ |Y| &\leq 4p(n - |X|)(n - |Y|), \\ (p+1)^2 |A|(n - 2|A|) &\leq 4p(n - |A|)2|A|. \end{aligned}$$

However, this implies

$$\begin{aligned} (p+1)^2 (n - 2|A|) &\leq 8p(n - |A|), \\ \text{i.e.,} \quad n((p+1)^2 - 8p) &\leq 2|A|((p+1)^2 - 4p) \\ &\leq 2|A|(p-1)^2 \\ &< 32pn \\ (p+1)^2 - 8p &< 32p \end{aligned}$$

which is impossible for $p \geq 41$.

Setting $p = 53$ (so that $X^{p,q} = X^{53,q}$ is regular of degree $p + 1 = 54$), we see that $X^{53,q}$ has $(1 + o(1))27n$ edges. This shows that Theorem 1 holds when $n = \frac{1}{2} q(q^2 - 1)$ for a prime $q \equiv 1 (\text{mod } 4)$.

If $\frac{1}{2} q_i (q_i^2 - 1) < n < \frac{1}{2} q_{i+1}(q_{i+1}^2 - 1) = n'$ where q_i and q_{i+1} are consecutive primes of the form $1 (\text{mod } 4)$, we can simply augment our initial set V to a slightly larger set V' of size n' by adding $n' - n = \delta(n)$ additional good elements. Standard results from number theory show that $\delta(n) = o(n^{3/5})$, for example. Since the Ramanujan graph query strategy of $\mathbf{Q}$ actually identifies $\Omega(n')$ balls of the majority color c from V' (for fixed p) then it certainly identifies a ball of the majority color of our original set V.

This proves Theorem 1 for all n. □

We remark that the constant 27 can be further reduced by using random sparse graphs and applying concentration estimates from probabilistic graph theory. However, such methods can only deduce the existence of a graph with the desired properties whereas we use an explicit construction (Ramanujan graphs) here.

3 Oblivious Strategies for the Plurality Problem

The *Plurality problem* generalizes the *Majority problem* where the goal is to identify a ball whose color occurs most often or show that there is no dominant color. When there are only $k = 2$ possible colors, the *Plurality problem* degenerates to the *Majority problem* with two colors, and hence there are tight bounds for both adaptive ($n - w_2(n)$) and oblivious ($n-2$ for n odd, $n-3$ for n even) strategies[9].

In general, it seems clear that the *k-color Plurality problem* should take more queries than the corresponding *Majority problem*. But exactly how much more difficult it is compared with the *Majority problem* was not so clear to us at the beginning. Similar arguments using concentration inequalities in random graphs seemed possible for achieving a linear upper bound. In the following section, we will prove the contrary by establishing a quadratic lower bound, even for the case when $k = 3$. Also note that the lower bound would remain quadratic even if we assume the existence of a plurality color through slight modification on the proof of Theorem 2. Intuitively, we can say that this is because the existence of a majority color gives us much more information than the existence of a plurality color.

3.1 Lower Bound

Theorem 2. *For the Plurality problem with $k = 3$ colors, the number of queries needed for any oblivious strategy satisfies*

$$PO_3(n) > \frac{n^2}{6} - \frac{3n}{2}$$

Proof. Consider any query graph G with n vertices and at most $\frac{n^2}{6} - \frac{3n}{2}$ edges. Therefore there must exist a vertex v with $deg(v) \leq n/3 - 3$. Denote the neighborhood of v by $N(v)$ (which consists of all vertices adjacent to v in G), and the remaining graph by $H = G \setminus (N(v) \cup \{v\})$. Hence,$H$ has at least $2n/3 + 2$ vertices.

Now split H into three parts H_1, H_2, and X where $|H_1| = |H_2| + 1$ and $|X| \leq 1$. Assign color 1 to all vertices in H_1, color 2 to all vertices in H_2, color 3 to all vertices in $N(v)$ and X, and color 1 or 2 to v. Note that based on either one of the two possible color assignments, all query answers are forced.

Since color 3 cannot possibly be the dominant color, we see that whether color 1 is the dominant color or there is no dominant color (because of a tie) solely depends on the color of v, which the Questioner cannot deduce from the query answers.

This proves the lower bound to the number of queries needed for the oblivious strategy is $\frac{n^2}{6} - \frac{3n}{2}$.

□

This quadratic lower bound also applies to all $k \geq 3$ colors for the *Plurality problem* using oblivious strategies, since we don't need to use any additional colors beyond 3 for this argument.

3.2 Upper Bound

A trivial upper bound is the maximum number of possible queries we can ask, which is $\binom{n}{2}$. In this section we will improve this by showing that for k colors, we have an upper bound of essentially $(1-1/k)\binom{n}{2}$ on $PO_k(n)$. This follows from the following fact.

Fact. Let $p \geq 1 - 1/k + \epsilon$ where $\epsilon > 0$. Then for n sufficiently large, a random graph G_p on n vertices almost surely has the property that for any subset S of vertices of size at least n/k, the graph $G_p[S]$ induced by S is connected.

Theorem 3. *For every* $\epsilon > 0$

$$PO_k(n) < (1 - 1/k + \epsilon)\binom{n}{2}$$

provided $n > n_0(\epsilon)$.

Proof. Given $p \geq 1 - 1/k + \epsilon$ where $\epsilon > 0$, we consider random graphs G_p for sufficiently large n (i.e. $n > n_0(\epsilon)$). Using the above fact, let $x_1 \in S$. Then almost surely, x_1 has a neighborhood of size at least $(1 - 1/k + \delta)n$ for some fixed $\delta > 0$. Thus, this neighborhood must intersect S, say in the point x_2. Now, look at the neighborhood of $x_1 \cup x_2$. Almost surely, this neighborhood has size at least $(1 - 1/k + \delta)n$, and so, hits $S \backslash \{x_1, x_2\}$, say in the point x_3. We can continue this process until we reach the set $X_t = \{x_1, x_2, \ldots, x_t\}$ where $t \approx log(n)$. It is clear (by the usual probabilistic arguments) that we can now continue this argument until the growing set X_t becomes all of S. As a result, using this graph, the Adversary cannot avoid forming a blue clique with all the vertices with the dominant color (since there are more than n/k of them). This proves Theorem 3. □

4 Adaptive Strategies for the Plurality Problem

Aigner et al.[2, 3] showed linear bounds for adaptive strategies for the *Plurality problem* with $k = 3$ colors. In this section, we first note a linear upper bound for general k in this case, and then strengthen it using a generalized argument.

Theorem 4. *For the Plurality problem with* k *colors where* $k \in Z^+$, *the minimum number of queries needed for any adaptive strategy satisfies*

$$PA_k(n) \leq (k-1)n - \frac{k(k-1)}{2}$$

Proof. There are k possible colors for the given n balls. We will use k buckets, each for a different possible color. All buckets are empty initially. The first ball s_1 is put in the first bucket b_1. The second ball is compared against a ball from b_1; if they have the same color, it is put in b_1, otherwise, it is put in a new

bucket b_2. Similarly, the i^{th} ball has to be compared against a ball from every non-empty bucket (at most $(i-1) \leq k-1$ many of them). Therefore the number of comparisons is no more than

$$1 + 2 + ... + (k-1) + (k-1)(n-k) = (k-1)n - \frac{k(k-1)}{2}$$

In [2], it was proved that $PA_3(n) \leq \frac{5}{3}n - 2$. We will now give a generalized proof for all $k \geq 3$ in this setting. It is also known to us through a very recent communication with the authors of [2] that they have independently come up with a different proof for similar claims which will appear in [3].

Theorem 5. *For the Plurality problem with k colors where $k \in Z^+$, the minimum number of queries needed for any adaptive strategy*

$$PA_k(n) \leq (k - \frac{1}{k} - 1)n - 2$$

Proof. Let us denote the comparison of ball a against b by $(a : b)$, and define a *color class* to be a set of balls having the same color. There are two phases in this game. Given n balls $\{s_1, s_2, ..., s_n\}$, in Phase I the Questioner handles one ball at a time (except for the first query) and keeps a state vector v_i after ball s_i is handled. Each v_i is simply the list of color class cardinalities, in non-decreasing order, $(a_{i1}, a_{i2}, ..., a_{ik})$ where $a_{i1} \geq a_{i2} ... \geq a_{ik}$. The Questioner also keeps a representative ball from every non-empty color class for comparisons and updates this list whenever there is a change in the state vector.
Claim: At every state, the Questioner has a strategy such that the total number t_i of comparisons up to v_i (inclusive) satisfies

$$t_i \leq (k-1)a_{i1} + (k-2)\sum_{j=2}^{k-1} a_{ij} + (k-1)a_{ik} - 2$$

Proof: We proceed by induction. After the first comparison, $v_2 = (1, 1, 0, ...)$ or $(2, 0, ...)$, so $t_2 = 1 \leq (k-1) + (k-2) - 2 \leq 2(k-1) - 2$ because $k \geq 3$.

For $2 \leq i \leq n$, let $v_i = (a_{i1}, a_{i2}, ..., a_{ik})$ be the state vector and $\{A_{i1}, A_{i2}, ..., A_{ik}\}$ be the set of corresponding representative balls (some may be null if the color class has cardinality 0). Now, ball s_{i+1} is to be handled as follows:

1. If there are no ties in the entries of v_i (i.e., $a_{is} \neq a_{it}$ for all s, t), we will compare s_{i+1} with the middle representative balls first, namely, the comparison order is

   $$(s_{i+1} : A_{i2}), (s_{i+1} : A_{i3}), ..., (s_{i+1} : A_{i(k-1)}), (s_{i+1} : A_1)$$

   with a total number of no more than $(k-1)$ comparisons. Note whenever the Adversary answers *Yes*, we know to which color class s_{i+1} belongs, and hence, we can skip the remaining comparisons.

2. Otherwise, pick any tie, say $a_{ij} = a_{i(j+1)}$, and compare s_{i+1} with all the other representative balls first, namely, the comparison order is

$$(s_{i+1} : A_{i1}), ..., (s_{i+1} : A_{i(j-1)}), (s_{i+1} : A_{i(j+2)}), ..., (s_{i+1} : A_{ik})$$

with a total number of no more than $(k-2)$ comparisons.

After identifying to which color class s_{i+1} belongs, only one of the numbers in v_i gets incremented by 1 and possibly moved forward, to maintain the non-decreasing order in v_{i+1}. Using the above strategy, we can ensure that no more than $(k-2)$ comparisons have been used in this round unless a_{i1} or a_{ik} gets incremented, in which case, their positions in the list do not change. Therefore, by the inductive hypothesis, we have

$$t_{i+1} \leq (k-1)a_{(i+1)1} + (k-2)\sum_{j=2}^{k-1} a_{(i+1)j} + (k-1)a_{(i+1)k} - 2$$

This proves the claim.

At state v_i, let r_i be the number of the remaining balls that have not been involved in any queries. Phase I ends when one of the following happens:

(A) $a_{i1} = a_{i2} = ... = a_{ik}$
(B) $r_i = a_{i1} - a_{i2} - 1$
(C) $r_i = a_{i1} - a_{i2}$

(Note that one of (A), (B), (C) will eventually occur.) To prove the theorem, we use induction where the cases for $n \leq 3$ are easy to verify. More comparisons may be needed in Phase II depending on in which case Phase I ends. If Phase I ends in case (A), we use the induction hypothesis; in case (B), no more comparisons are needed because A_{i1} is a Plurality ball; in case (C), we need no more than r_i more comparisons to identify A_{i1} or A_{i2} as a Plurality ball. In all cases, we can show (using the claim) with arguments similar to those in [2] that

$$PA_k(n) \leq (k-1)n - n/k - 2 = (k - \frac{1}{k} - 1)n - 2$$

This proves the theorem. □

5 Conclusion

In this paper, we established upper and lower bounds for oblivious and adaptive strategies used to solve the *Majority* and *Plurality problems*. These problems originally arose from applications in fault tolerant system design. However, the interactive nature of these problems also places them in the general area of interactive computing. It is therefore desirable to develop more techniques to solve this type of problems efficiently as well as to understand the limits of our ability in doing so.

References

1. M. Aigner, "Variants of the Majority Problem", *Applied Discrete Mathematics,* **137***, (2004), 3-25.*
2. M. Aigner, G. De Marco, M. Montangero, "The Plurality Problem with Three Colors", *STACS 2004, 513-521.*
3. M. Aigner, G. De Marco, M. Montangero, "The Plurality Problem with Three Colors and More", *Theoretical Computer Science, to appear, (2005).*
4. N. Alon, "Eigenvalues and Expanders", *Combinatorica* **6** *(1986), 86-96.*
5. L. Alonso, E. Reingold, R. Schott, "Average-case Complexity of Determining the Majority", *SIAM J. Computing* **26***, (1997), 1-14.*
6. P. M. Blecher, "On a Logical Problem", *Discrete Mathematics* **43***, (1983), 107-110*
7. B. Bollobás, "Random graphs", 2nd ed., *Cambridge Studies in Advanced Mathematics, 73. Cambridge University Press, Cambridge, 2001*
8. F. R. K. Chung, "Spectral Graph Theory", *CBMS Lecture Notes, AMS Publications, 1997.*
9. F. R. K. Chung, R. L. Graham, J. Mao, and A. C. Yao, "Finding Favorites", *Electronic Colloquium on Computational Complexity(ECCC) (078): 2003*
10. M. J. Fischer and S. L. Salzberg, "Finding a Majority among n Votes", *J. Algorithms 3: 375-379 (1982).*
11. D. E. Knuth, *personal communication.*
12. D. E. Knuth, "The Art of Computer Programming, Volume 3. Sorting and Searching", *Addison-Wesley Publishing Co., Reading, Mass., 1973.*
13. A. Lubotzky, R. Phillips and P. Sarnak, "Ramanujan Graphs", *Combinatorica* **8** *(1988), 261-277.*
14. J. Moore, "Proposed problem 81-5", *J. Algorithms 2: 208-210 (1981).*
15. M. Saks and M. Werman, "On Computing Majority by Comparisons", *Combinatorica* **11***(4) (1991), 383-387.*
16. A. Taylor and W. Zwicker, *personal communication.*
17. G. Wiener, "Search for a Majority Element", *J. Statistical Planning and Inference 100 (2002), 313-318.*

A Note on Zero Error Algorithms Having Oracle Access to One NP Query*

Jin-Yi Cai[1,2] and Venkatesan T. Chakaravarthy[3,**]

[1] Computer Science Department, Univ. of Wisconsin, Madison, WI 53726, USA
jyc@cs.wisc.edu
[2] Tsinghua University, Beijing, China
[3] IBM India Research Lab, IIT Campus, New Delhi 110016, India
vechakra@in.ibm.com

Abstract. It is known that $S_2^p \subseteq ZPP^{NP}$ [3]. The reverse direction of whether ZPP^{NP} is contained in S_2^p remains open. We show that if the zero-error algorithm is allowed to ask only one query to the NP oracle (for any input and random string), then it can be simulated in S_2^p. That is, we prove that $ZPP^{NP[1]} \subseteq S_2^p$.

1 Introduction

Symmetric alternation, or the class S_2^p, was independently introduced by Russell and Sundaram [15] and Canetti [5]. It is a proof system in which a polynomial time verifier ascertains whether or not an input string x belongs to a language L by interacting with two computationally all powerful provers. The two provers, called the YES-PROVER and the NO-PROVER, make contradictory claims: $x \in L$ and $x \notin L$, respectively. Of course, only one of them could be honest. In order to substantiate their claims, the two competing provers provide polynomially long strings y and z, respectively, as certificates. The verifier analyzes the input x and the certificates y and z, and votes in favor of one of the provers. We require that if $x \in L$, then there exists a certificate y using which the YES-PROVER can win the vote, for any certificate z provided by the NO-PROVER. Similarly, if $x \notin L$, then there should exist a z using which the NO-PROVER can win the vote, for any certificate y provided by the YES-PROVER. We call the certificates satisfying the above requirements as *irrefutable certificates*. We can rephrase the requirements as follows. If $x \in L$, then the YES-PROVER has an irrefutable certificate, and if $x \notin L$, then the NO-PROVER has an irrefutable certificate. The symmetric alternation class S_2^p consists of languages having a proof system of the above type. We will formally define S_2^p later.

Russell and Sundaram [15] and independently Canetti [5] introduced the class S_2^p and showed that $BPP \subseteq S_2^p$. As it is easy to show $S_2^p \subseteq \Sigma_2^p \cap \Pi_2^p$, the above result improves the classical Sipser–Lautemann theorem that $BPP \subseteq \Sigma_2^p \cap \Pi_2^p$ [12, 17]. Obtaining this improvement was one of their motivations in

* Research supported in part by NSF grant CCR-0208013.
** Research conducted while the author was at the Univesity of Wisconsin

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 339–348, 2005.

introducing S_2^p. It was also shown in [15] that S_2^p contains MA and P^{NP}. Symmetric alternation has been gaining attention recently and several nice results involving the class are known (see [3, 4, 6–8, 16]).

S_2^p shares some nice properties with ZPP^{NP}. For instance, both contain P^{NP} and MA, and both are contained in $\Sigma_2^p \cap \Pi_2^p$. Based on this observation, Goldreich and Zuckerman [8] raised the relationship between the two as an open question. Cai [3] answered the question in part by showing that $S_2^p \subseteq ZPP^{NP}$. Figure 1 illustrates the relationships among S_2^p and other classes.

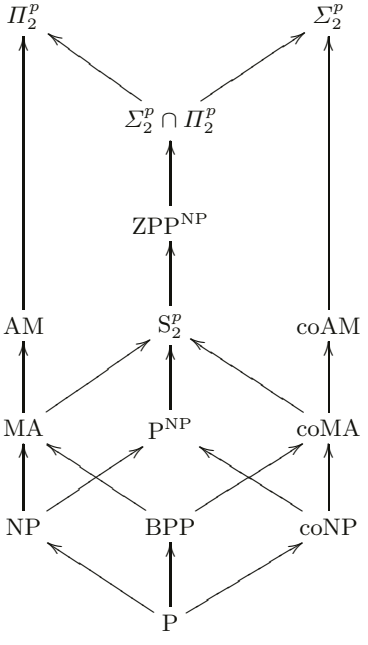


Fig. 1. S_2^p and other classes

As discussed above, we have that $P^{NP} \subseteq S_2^p \subseteq ZPP^{NP}$ [3, 15]. The reverse containments remain challenging open problems, but they can be obtained under some widely-believed complexity theoretic assumptions. A long line of research has demonstrated that randomized computations can be simulated deterministically for broad classes of problems under plausible assumptions. Working in this framework, Klivans and van Melkebeek [11] built on a famous result of Impagliazzo and Wigderson [10] and showed that if there is a language $L \in NE \cap coNE$ such that any family of circuits with SAT oracle gates computing L is of size $2^{\Omega(n)}$ then $BPP^{NP} = P^{NP}$. Clearly, the same assumption also implies that $ZPP^{NP} = S_2^p = P^{NP}$. On the other hand, Shaltiel and Umans [16] showed that a weaker assumption is sufficient to obtain $S_2^p = P^{NP}$. Their assumption involves co-nondeterministic circuits, instead of circuits with SAT gates and it is the one used in [13] to derandomize AM.

The above discussion shows that $P^{NP} = S_2^p = ZPP^{NP}$ under some plausible assumptions. However, it remains a challenging open problem to derive the above result unconditionally. In this paper, we focus on the issue of whether ZPP^{NP} is contained in S_2^p and prove a weak result along these lines. Recall that $L \in ZPP^{NP}$ means that the language is computed by a zero-error polynomial time algorithm having oracle access to a NP language. Such an algorithm may ask polynomially many queries to the oracle. We consider a subset of these algorithms that make only one oracle query (for any input and any random string). Our main result is that such algorithms can be simulated in S_2^p. That is, we prove that $ZPP^{NP[1]} \subseteq S_2^p$.

We note that whether $ZPP^{NP[2]}$ is contained in S_2^p remains open. The class $ZPP^{NP[2]}$ contains the class $AM \cap coAM$, which in turn contains ZPP^{GI}, where GI is the graph isomorphism problem. Even determining whether $AM \cap coAM$ or ZPP^{GI} is contained in S_2^p remain interesting open problems.

Our result that $ZPP^{NP[1]} \subseteq S_2^p$ is tight in the following sense. We consider the containment of $ZPP^{NP[1]}$ in other important classes in the second level of the polynomial time hierarchy. It is unlikely that $ZPP^{NP[1]}$ is contained in AM, since $coNP \subseteq ZPP^{NP[1]}$ and it is shown in [2] that $coNP \subseteq AM$ implies PH collapses to AM. Consequently, $ZPP^{NP[1]}$ is unlikely to be contained in smaller classes such as MA and NP. One potential candidate that might contain $ZPP^{NP[1]}$ is P^{NP}. As remarked before, under some widely-believed hypothesis, $ZPP^{NP[1]}$ (or more generally, BPP^{NP}) is contained in P^{NP} [11]. But, it is difficult to prove the above claim unconditionally, since $BPP \subseteq ZPP^{NP[1]}$ and the containment of BPP in P^{NP} has been a long standing open problem.

The fact that $BPP \subseteq ZPP^{NP[1]}$ can be shown in many ways. Nisan and Wigderson [14] and Goldreich and Zuckerman [8] present alternative proofs for $BPP \subseteq ZPP^{NP}$. The former proof is based on Nisan-Wigderson pseudorandom generator [14] and the latter uses extractor based amplification [18] of BPP. These proofs indeed show that $BPP \subseteq ZPP^{NP[1]}$. But, they use the above-mentioned heavy tools. We observe that the result can be obtained through a small modification of a standard proof of the fact $BPP \subseteq ZPP^{NP}$. We also present the details of this observation.

We now sketch the main idea of the proof of $ZPP^{NP[1]} \subseteq S_2^p$. Since the satisfiability problem SAT is NP-Complete, it suffices to show that $ZPP^{SAT[1]} \subseteq S_2^p$. Because the underlying computation is of the ZPP type, it is easy to handle an input string x, if there is some random string y such that M accepts or rejects x, i.e., M reaches a definitive answer, *and* the oracle query φ asked by M along the computational path specified by y is satisfiable. In this case, the honest prover has an irrefutable certificate consisting of the string y and a satisfying truth assignment of φ. The harder part of the proof involves handling input strings that do not satisfy the above property. Let x be such an input string. We consider the following (possibly incorrect) simulation of M that does not use the SAT oracle: given strings x and y, run the machine M with x as input and y as the random string, assuming answer to the oracle query is "No". The main observation is that the set of all accepting paths in this simulation is

"large" if and only if $x \in L$. In our proof system, the YES-PROVER (similarly, the NO-PROVER) would try to prove that $x \in L$ ($x \notin L$) by showing that the above set is large (small). This is achieved by using ideas from the proof of BPP $\subseteq \mathrm{S}_2^p$.

Organization. The paper is organized as follows. Section 2 provides the necessary definitions. The main result is proved in Section 3. Section 4 presents the details of our observation that BPP $\subseteq \mathrm{ZPP}^{\mathrm{NP}[1]}$. We conclude the paper by stating some open problems in Section 5.

2 Preliminaries

We start with the formal definition of S_2^p.

Definition 1 ([5, 15]). *A language L is in S_2^p if there exists a polynomial-time computable boolean predicate $P(\cdot,\cdot,\cdot)$ and a polynomial p such that, for all x,*

1. $x \in L \iff (\exists^{p(|x|)} y)(\forall^{p(|x|)} z)[P(x,y,z) = 1]$, *and*
2. $x \notin L \iff (\exists^{p(|x|)} z)(\forall^{p(|x|)} y)[P(x,y,z) = 0]$.

Our next task is to define $\mathrm{ZPP}^{\mathrm{NP}[1]}$. We first consider an issue pertaining to its definition. The usual approaches for amplifying the success probability of randomized computations involve running it multiple times using strings chosen independently at random (or according to some suitable distribution). In the case of a $\mathrm{ZPP}^{\mathrm{NP}[1]}$ algorithms, this approach involves asking multiple queries to the NP oracle. It is not clear whether the success probability $\mathrm{ZPP}^{\mathrm{NP}[1]}$ algorithms can be amplified asking only one query. In general, the issue arises in the context of randomized bounded query computations. It seems to be an interesting problem worthy of further investigation. In our discussion, we circumvent the issue by defining $\mathrm{ZPP}^{\mathrm{NP}[1]}$ in a way that would make our $\mathrm{ZPP}^{\mathrm{NP}[1]} \subseteq \mathrm{S}_2^p$ result stronger: we require the success probability to be only $1/2 + 1/poly(n)$. It remains open whether $\mathrm{ZPP}^{\mathrm{NP}[1]}$ machines with success probabilities less than $1/2$ can be simulated in S_2^p.

Definition 2. *We say that a language $L \in \mathrm{ZPP}^{\mathrm{NP}[1]}$, if there exists a randomized oracle Turing machine $M(\cdot,\cdot)$, a language $A \in \mathrm{NP}$ and a polynomial $p(\cdot)$ such that, given oracle access to A, the machine M satisfies the following requirements for any input string $x \in \{0,1\}^n$.*

- *On any random string, M asks at most one query to the oracle A.*
- *The machine has zero-error. Meaning, for any random string, the output belongs to $\{$ACCEPT , REJECT , ?$\}$, such that if $x \in L$ then the output is either* ACCEPT *or ?, and if $x \notin L$ then the output is either* REJECT *or ?.*
- *The machine has good success probability:*

$$\Pr[M \text{ outputs ACCEPT or REJECT }] \geq \frac{1}{2} + \frac{1}{p(n)},$$

where the probability is taken over the random coin tosses of M.

3 $\mathbf{ZPP^{NP[1]} \subseteq S_2^p}$

In this section, we prove our main result. As outlined in the introduction, the proof involves considering two types of input strings. The first consists of input strings x such that for some random string y, the zero-error algorithm outputs ACCEPT or REJECT and the query φ asked by the algorithm is satisfiable. The second type consists of inputs strings that do not satisfy the above property. It is easy to handle the first type of strings. For the second type of strings, we consider an incorrect simulation of the algorithm that assumes the answer to the query is "No". Under this simulation, the set of accepting paths is "large" if and only if x belongs to the language. We make use of a standard lemma regarding "large" and "small" sets for handling the second type of input strings. The following notation is used in the lemma and our proof.

For two m-bit strings $u = u_1u_2 \dots u_m$ and $v = v_1v_2 \dots v_m$, let $u \oplus v$ denote the bitwise XOR of u and v: $u \oplus v = u_1 \oplus v_1 \dots u_m \oplus v_m$. For a set $S \subseteq \{0,1\}^m$, let $S \oplus u$ be the set $\{s \oplus u | s \in S\}$. We call $S \oplus u$ the shift of S by u.

Suppose a set $S \subseteq \{0,1\}^m$ is sufficiently large. Then, if we choose a suitable number of shift vectors $s_1, s_2, \dots, s_k$ at random, with high probability, the union of these shifts will "cover" the entire space: $\bigcup_{i=1}^{k}(S \oplus u_i) = \{0,1\}^m$. On the other hand, if S is a sufficiently small and k is small enough, then for any set of vectors $s_1, s_2, \dots, s_k$, the shifts will cover only a constant fraction of $\{0,1\}^m$. The following lemma formalizes the above discussion.

Lemma 1. *[12] Let $S \subseteq \{0,1\}^m$ and $k = \lfloor \frac{m}{2} \rfloor$.*

1. *If $S/2^m \geq 1 - 1/m$ then*

$$\Pr_{s_1,s_2,\dots,s_k \in \{0,1\}^m}\left[\bigcup_{i=1}^{k}(S \oplus s_i) = \{0,1\}^m\right] \geq 1 - \frac{2^m}{m^k}.$$

2. *If $S/2^m < 1/m$ then for any $s_1, s_2, \dots, s_k \in \{0,1\}^m$,*

$$\Pr_{u \in \{0,1\}^m}\left[u \in \bigcup_{i=1}^{k}(S \oplus s_i)\right] \leq \frac{1}{2}.$$

Theorem 1. $\mathrm{ZPP}^{\mathrm{NP[1]}} \subseteq \mathrm{S}_2^p$.

Proof. Since the satisfiability problem (SAT) is NP-complete, it suffices to show that $\mathrm{ZPP}^{\mathrm{SAT[1]}} \subseteq \mathrm{S}_2^p$. Let $L \in \mathrm{ZPP}^{\mathrm{SAT[1]}}$ via a randomized oracle Turing machine M. Let $r(n)$ be the number of random bits used by M on an input of length n, where $r(\cdot)$ is a polynomial. Consider running the machine M with some $x \in \{0,1\}^n$ as the input and some $y \in \{0,1\}^{r(n)}$ as the random string. Without loss of generality, we assume that the machine asks a query to the SAT oracle and let $\varphi_{x,y}$ denote that query. Let $M(x,y)$ be the output of the machine when the query is answered correctly. Let $M_{no}(x,y)$ denote the output when the answer to the query is taken as "No" (which is an incorrect answer, if $\varphi_{x,y}$ is satisfiable). $M_{yes}(x,y)$ is defined similarly.

Fix an input string $x \in \{0,1\}^n$ and write $r = r(n)$. We call x *nice* if there exists $y \in \{0,1\}^r$ such that $\varphi_{x,y} \in$ SAT and $M(x,y)$ is either ACCEPT or REJECT . Define

$$E_x = \{y \mid y \in \{0,1\}^r \text{ and } M_{no}(x,y) = \text{ACCEPT }\}.$$

We observe that, if x is not nice, then the following two implications are true.

$$x \in L \Longrightarrow \frac{|E_x|}{2^r} \geq \frac{1}{2} + \frac{1}{p(n)}$$
$$x \notin L \Longrightarrow \frac{|E_x|}{2^r} \leq \frac{1}{2} - \frac{1}{p(n)}$$

We wish to amplify the gap in the size of E_x in the two cases and we do it in a standard way. Write $t = (p(n))^2$ and $m = t \cdot r$. Define a set $\widetilde{E}_x \subseteq \{0,1\}^m$ as follows.

$$\widetilde{E}_x = \{y_1 \circ y_2 \circ \cdots \circ y_t | \text{ majority of } y_1, y_2, \ldots, y_t \text{ belong to the set } E_x\}$$

By Chernoff bounds, if x is not nice then we have the following two implications.

$$x \in L \Longrightarrow \frac{|\widetilde{E}_x|}{2^m} \geq 1 - \frac{1}{m}$$
$$x \notin L \Longrightarrow \frac{|\widetilde{E}_x|}{2^m} \leq \frac{1}{m}$$

We now present our proof system. Let $x \in \{0,1\}^n$ be the input string. The YES-PROVER's certificate consists of two components. The first component is a single bit using which he makes his claim on whether x is nice or not. If he claims it to be nice, then the second component should consist of a string $y \in \{0,1\}^r$ and a truth assignment σ to the formula $\varphi_{x,y}$. (We expect that $M(x,y) =$ ACCEPT , $\varphi_{x,y} \in$ SAT and σ is a satisfying truth assignment of $\varphi_{x,y}$, proving that $x \in L$. Of course, the prover could be lying!) If he claims x is not nice, then the second component should be sequence of k strings $\overrightarrow{s} = (s_1, s_2, \ldots, s_k) \in (\{0,1\}^m)^k$, where $k = \lfloor \frac{m}{2} \rfloor$. (The expectation is that the union of the shifts of $\widetilde{E}_x$ by these strings covers the entire space). The NO-PROVER's certificate is similar. Its first component is a bit using which the NO-PROVER makes his claim on whether x is nice or not. If he claims it to be nice, then the second component should consist of a string $y \in \{0,1\}^r$ and a truth assignment σ to the formula $\varphi_{x,y}$. If he claims x is not nice, then the second component should consist of a sequence of ℓ strings $\overrightarrow{u} = (u_1, u_2, \ldots, u_\ell) \in (\{0,1\}^m)^\ell$, where $\ell = km + 1$. (The expectation is that for any set of strings $s_1, s_2, \ldots, s_k$, there is at least one u_i that is not covered by these shifts).

The verifier's algorithm works as follows. If the YES-PROVER claims that x is nice, the verifier checks whether the string y and σ provided by him satisfy the conditions $\varphi_{x,y}(\sigma) = 1$ and $M_{yes}(x,y) =$ ACCEPT . If so, he votes in favor of the YES-PROVER, else he votes in favor of the NO-PROVER. Suppose the

YES-PROVER claims x is not nice. If the NO-PROVER claims x is nice, then the verifier checks whether the string y and σ provided by the NO-PROVER satisfy the conditions $\varphi_{x,y}(\sigma) = 1$ and $M_{yes}(x, y) =$ ACCEPT . If so, he votes in favor of the NO-PROVER, else he votes in favor of the YES-PROVER.

The interesting case is where both provers claim that x is not nice. Here we consider the second components of the certificates $\overrightarrow{s} = (s_1, s_2, \ldots, s_k)$ and $\overrightarrow{u} = (u_1, u_2, \ldots, u_\ell)$, provided by the YES-PROVER and NO-PROVER, respectively. We say that $\overrightarrow{s}$ *beats* $\overrightarrow{u}$, if

$$\forall(1 \leq j \leq \ell) \left[u_j \in \bigcup_{i=1}^{k} \widetilde{E}_x \oplus s_i \right]. \tag{1}$$

If the above predicate is false, then we say that $\overrightarrow{u}$ *beats* $\overrightarrow{s}$. Given $\overrightarrow{s}$ and $\overrightarrow{u}$, if $\overrightarrow{s}$ beats $\overrightarrow{u}$, then the verifier votes in favor of the YES-PROVER and otherwise, he votes in favor of the NO-PROVER.

Notice that

$$u_j \in \widetilde{E}_x \oplus s_i \iff u_j \oplus s_i \in \widetilde{E}_x.$$

Checking whether a given string belongs to $\widetilde{E}_x$ involves membership testing in E_x, which can be carried out in polynomial time. Thus the verifier's algorithm runs in polynomial time.

We next argue the correctness of the proof system. It is clear that the honest prover has an irrefutable certificate, if the input string x is nice. The interesting case is where x is not nice. Suppose $x \in L$ and we will show that the YES-PROVER has an irrefutable certificate. In this scenario, since $|\widetilde{E}_x|/2^m \geq 1 - 1/m$, by Lemma 1, there exists $\overrightarrow{s} = (s_1, s_2, \ldots, s_k)$ such that $\bigcup_{i=1}^{k}(S \oplus u_i) = \{0,1\}^m$. It follows that $\overrightarrow{s}$ beats any $\overrightarrow{u} = (u_1, u_2, \ldots, u_\ell)$. So the YES-PROVER has an irrefutable certificate. Now suppose $x \notin L$. Here $|\widetilde{E}_x|/2^m \leq 1/m$ and hence, by Lemma 1, for any $\overrightarrow{s} = (s_1, s_2, \ldots, s_k)$,

$$\Pr_{u \in \{0,1\}^m} \left[u \in \bigcup_{i=1}^{k} (\widetilde{E}_x \oplus u_i) \right] \leq 1/2.$$

It follows that, choosing $u_1, u_2, \ldots, u_\ell$ independently at random,

$$\Pr_{\overrightarrow{u} = (u_1, u_2, \ldots, u_\ell)} [\overrightarrow{s} \text{ beats } \overrightarrow{u}] \leq \frac{1}{2^\ell}.$$

As a consequence,

$$\Pr_{\overrightarrow{u} = (u_1, u_2, \ldots, u_\ell)} \left[\exists \overrightarrow{s} \in \{0,1\}^{km} (\overrightarrow{s} \text{ beats } \overrightarrow{u}) \right] \leq \frac{2^{km}}{2^\ell} \leq \frac{1}{2},$$

where the last inequality follows from our choice of ℓ. We conclude that there exists some $\overrightarrow{u}$ that beats all $\overrightarrow{s} \in \{0,1\}^{km}$ and hence, the NO-PROVER has an irrefutable certificate. □

4 BPP $\subseteq$ ZPP$^{\mathrm{NP[1]}}$: A Simple Proof

A standard way of proving BPP $\subseteq$ ZPP$^{\mathrm{NP}}$ (via Lemma 1) involves asking two oracle queries. We observe that the number of queries can be reduced to one via a small modification to the construction. This observation leads to a simple proof of BPP $\subseteq$ ZPP$^{\mathrm{NP[1]}}$. We note that the result can also be obtained by other means. For instance, Nisan and Wigderson [14] and Goldreich and Zuckerman [8] present alternative proofs for BPP $\subseteq$ ZPP$^{\mathrm{NP}}$, by using Nisan-Wigderson pseudo-random generator [14] and extractor based amplification [18], respectively. The constructions in these proofs, in fact, make use of only one query.

Theorem 2. BPP $\subseteq$ ZPP$^{\mathrm{NP[1]}}$.

Proof. One standard way of proving BPP $\subseteq$ ZPP$^{\mathrm{NP}}$ uses Lemma 1 and it goes as follows. Let $L \in$ BPP and M be a randomized algorithm for L. Let x be an input string. Via suitable amplification, we assume that M has error probability $\leq 1/m$, where m is the number of coin tosses of M on input x. Let $E_x = \{y | M(x, y) = 1\}$. Given the input x, our ZPP$^{\mathrm{NP}}$ simulation chooses $s_1, s_2, \ldots, s_m \in \{0,1\}^m$ at random and asks two queries to NP oracle:

$$(Q_1) \quad \bigcup_{i=1}^{m} [E_x \oplus s_i] = \{0,1\}^m$$

$$(Q_2) \quad \bigcap_{i=1}^{m} [E_x \oplus s_i] = \phi$$

If the answer to Q_1 is "yes", then the simulation outputs ACCEPT . If the answer to Q_2 is "yes" then it outputs REJECT . If both the answers are "no" it outputs "?". (It is not possible to have "yes" answers to both questions). Lemma 1 guarantees that the simulation is correct and that it has high probability of success.

The above simulation uses two queries. We observe that the number of queries can be reduced to one, using the following simple idea. We first simulate the machine M, using a randomly chosen string. The outcome of the machine gives us a clue as to whether or not $x \in L$, which is correct with high probability. Then we take only one of the two paths of the previous ZPP$^{\mathrm{NP}}$ simulation, where the path is chosen according to the outcome. Formally, if $M(x, y) = 1$, then we ask the query Q_1. If the oracle answers "yes", then output ACCEPT , else output "?". On the other hand, if $M(x, y) = 0$, then we ask the query Q_2. If the oracle answers "yes", then output REJECT , else "?". It is easy to argue that this algorithm is correct and that it has high probability of success. □

5 Open Problems

As we discussed before, it is known that P$^{\mathrm{NP}} \subseteq \mathrm{S}_2^p \subseteq$ ZPP$^{\mathrm{NP}}$[3, 15]. The reverse containments are challenging open problems. But, we can prove them under

some complexity theoretic assumptions. Under a plausible hypothesis, we know that $\mathrm{ZPP}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}}$ [11]. In a recent work, Shaltiel and Umans [16] present a conditional derandomization of Cai's simulation [3] of S_2^p in $\mathrm{ZPP}^{\mathrm{NP}}$. They show that a hypothesis weaker than the one used in [11] suffices to obtain $\mathrm{S}_2^p = \mathrm{P}^{\mathrm{NP}}$. Their hypothesis is the one used in [13] to show AM = NP. In a similar spirit, it would be nice if we could weaken the assumptions needed for showing $\mathrm{ZPP}^{\mathrm{NP}} \subseteq \mathrm{S}_2^p$.

It is also worthwhile to consider classes contained in $\mathrm{ZPP}^{\mathrm{NP}}$ and show them to be in S_2^p. We proved that $\mathrm{ZPP}^{\mathrm{NP}[1]}$ is in S_2^p, but we required that the randomized algorithms have success probability greater than half. It would be nice to show the same result when the success probability is less than half (say, 1/4). Whether $\mathrm{ZPP}^{\mathrm{NP}[2]}$ (even with success probability close to one) is contained in S_2^p also remains open. AM∩coAM is another interesting class contained in $\mathrm{ZPP}^{\mathrm{NP}}$ that is not known to be in S_2^p. The above questions are related and are in the decreasing order of difficulty. First of all, it is not hard to show that $\mathrm{AM} \cap \mathrm{coAM} \subseteq \mathrm{ZPP}^{\mathrm{NP}[2]}$. Secondly, we can easily simulate two queries to an NP oracle with only one query incurring a loss in the success probability by a factor of two. Another open question along these lines is the containment of $\mathrm{ZPP}^{\mathrm{GI}}$ in S_2^p, where GI refers to the graph isomorphism problem. This might be easier to prove, since $\mathrm{ZPP}^{\mathrm{GI}} \subseteq \mathrm{AM} \cap \mathrm{coAM}$ [1, 9].

References

1. L. Babai and S. Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.
2. R. Boppana, J. Håstad, and S. Zachos. Does Co-NP have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987.
3. J. Cai. $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, 2001.
4. J. Cai, V. Chakaravarthy, L. Hemaspaandra, and M. Ogihara. Competing provers yield improved Karp–Lipton collapse results. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science*, 2003.
5. R. Canetti. More on BPP and the polynomial-time hierarchy. *Information Processing Letters*, 57(5):237–241, 1996.
6. L. Fortnow, R. Impagliazzo, V. Kabanets, and C. Umans. On the complexity of succinct zero-sum games. Technical Report TR04–001, Electronic Colloquium on Computational Complexity, 2004. Available at http://www.eccc.uni-trier.de/eccc.
7. L. Fortnow, A. Pavan, and S. Sengupta. Proving SAT does not have small circuits with an application to the two queries problem. In *Proceedings of the 18th Annual IEEE Conference on Computational Complexity*, 2003.
8. O. Goldreich and D. Zuckerman. Another proof that BPP ⊆ PH (and more). Technical Report TR97–045, Electronic Colloquium on Computational Complexity, 1997. Available at http://www.eccc.uni-trier.de/eccc.
9. S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the 18th ACM Symposium on Theory of Computing*, 1986.

10. R. Impagliazzo and A. Wigderson. P=BPP unless E has subexponential circuits: Derandomizing the XOR lemma. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, 1997.
11. A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002.
12. C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1982.
13. P. Miltersen and N. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, 1999.
14. N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994.
15. A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Computational Complexity*, 7(2):152–162, 1998.
16. R. Shaltiel and C. Umans. Pseudorandomness for approximate counting and sampling. Technical Report TR04–086, Electronic Colloquium on Computational Complexity, 2004. Available at http://www.eccc.uni-trier.de/eccc.
17. M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, 1983.
18. D. Zuckerman. Simulating BPP using a general weak random source. *Algorithmica*, 16(4/5):367–391, 1996.

On the Complexity of Computing the Logarithm and Square Root Functions on a Complex Domain*

Ker-I Ko and Fuxiang Yu

Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794, USA
{keriko,fuxiang}@cs.sunysb.edu

Abstract. The problems of computing single-valued, analytic branches of the logarithm and square root functions on a bounded, simply connected domain S are studied. If the boundary ∂S of S is a polynomial-time computable Jordan curve, the complexity of these problems can be characterized by counting classes $\#P$, MP (or $MidBitP$), and $\oplus P$: The logarithm problem is polynomial-time solvable if and only if $FP = \#P$. For the square root problem, it has been shown to have the upper bound P^{MP} and lower bound $P^{\oplus P}$. That is, if $P = MP$ then the square root problem is polynomial-time solvable, and if $P \neq \oplus P$ then the square root problem is not polynomial-time solvable.

1 Introduction

Let S and T be two bounded, simply connected domains in the two-dimensional plane such that S and T are disjoint (i.e., $S \cap T = \emptyset$). The logarithm function defined on $S \times T$ is the multi-valued function $\log(\mathbf{z} - \mathbf{a})$ that satisfies $e^{\log(\mathbf{z}-\mathbf{a})} = \mathbf{z} - \mathbf{a}$ for $\mathbf{z} \in S$ and $\mathbf{a} \in T$. It is well known that the real part of $\log(\mathbf{z} - \mathbf{a})$ is always $\log|\mathbf{z} - \mathbf{a}|$. On the other hand, $\log(\mathbf{z} - \mathbf{a})$ has infinitely many single-valued, analytic branches; furthermore, for a fixed pair $\langle \mathbf{z}_0, \mathbf{a}_0 \rangle \in S \times T$, $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a}_0)$ remains the same for an arbitrary single-valued, analytic branch f_1 of $\log(\mathbf{z} - \mathbf{a})$ (cf. Henrici [8]). Therefore, to compute all single-valued, analytic branches of $\log(\mathbf{z} - \mathbf{a})$, we only need to compute $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a}_0)$ for an arbitrary single-valued, analytic branch f_1 of $\log(\mathbf{z} - \mathbf{a})$. However, in general, it is still not an easy task. For example, even if $\mathbf{z} - \mathbf{a}_0$ and $\mathbf{z}_0 - \mathbf{a}_0$ have the same argument(i.e., there exists an angle $\theta \in [0, 2\pi)$ such that $\mathbf{z} - \mathbf{a}_0 = |z - \mathbf{a}_0|e^{i\theta}$ and $\mathbf{z}_0 - \mathbf{a}_0 = |\mathbf{z}_0 - \mathbf{a}_0|e^{i\theta}$), the imaginary part of $f_1(\mathbf{z} - \mathbf{a}_0) - f_1(\mathbf{z}_0 - \mathbf{a}_0)$ is not necessarily zero. Indeed, this imaginary part depends on how many times a path in S from $\mathbf{z}_0$ to $\mathbf{z}$ must wind around the point $\mathbf{a}_0$. This suggests that the computational complexity of finding single-valued, analytic branches

* This material is based upon work supported by National Science Foundation under grant No. 0430124.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 349–358, 2005.

of $\log(\mathbf{z}-\mathbf{a})$ depends on the complexity of the domains S and T. We study this problem and the related square root problem of finding the single-valued, analytic branches of $\sqrt{\mathbf{z}-\mathbf{a}}$, in the context of complexity theory of real functions of Ko and Friedman [11]. In this model, we use the oracle Turing machine as the basic computational model, and define the complexity of a real function in terms of precision of the output values of the functions under consideration.

Note that since $f_1(\mathbf{z}-\mathbf{a})-f_1(\mathbf{z}_0-\mathbf{a}_0)=(f_1(\mathbf{z}-\mathbf{a})-f_1(\mathbf{z}_0-\mathbf{a}))-(f_1(\mathbf{z}_0-\mathbf{a})-f(\mathbf{z}_0-\mathbf{a}_0))$, as far as the computational complexity of the single-valued, analytic branches of $\log(\mathbf{z}-\mathbf{a})$ is concerned, by symmetry, we only need to study how to compute $f_1(\mathbf{z}-\mathbf{a})-f_1(\mathbf{z}_0-\mathbf{a})$. That is, there is only one bounded, simply connected domain S under consideration. Moreover, we assume that $\mathbf{a}$ is not on the boundary ∂S of S, and $\mathbf{z}_0$ is allowed to be on ∂S. We also assume that the boundary ∂S of S is a polynomial-time computable Jordan curve (see the definition in Section 2). Under these settings, we consider the following problem. (In the following, we let $S-\mathbf{a}$ denote the domain $\{\mathbf{w}-\mathbf{a} \mid \mathbf{w}\in S\}$.)

> Logarithm Problem: Let S be a bounded, simply connected domain whose boundary ∂S is a polynomial-time computable Jordan curve. Let $\mathbf{z}_0$ be a fixed point in $S\cup\partial S$. Given two points $\mathbf{z}\in S$ and $\mathbf{a}\in\mathbb{C}-(S\cup\partial S)$, compute $f_1(\mathbf{z}-\mathbf{a})-f_1(\mathbf{z}_0-\mathbf{a})$, where f_1 is an arbitrary single-valued, analytic branch of $\log\mathbf{z}$ on domain $S-\mathbf{a}$.

Similarly, the problem of computing single-valued, analytic branches of $\sqrt{\mathbf{z}-\mathbf{a}}$ is equivalent to the following problem.

> Square Root Problem: Let S be a bounded, simply connected domain whose boundary ∂S is a polynomial-time computable Jordan curve. Let $\mathbf{z}_0$ be a fixed point in $S\cup\partial S$. Given two points $\mathbf{z}\in S$ and $\mathbf{a}\in\mathbb{C}-(S\cup\partial S)$, compute $f_1(\mathbf{z}-\mathbf{a})/f_1(\mathbf{z}_0-\mathbf{a})$, where f_1 is an arbitrary single-valued, analytic branch of $\sqrt{\mathbf{z}}$ on domain $S-\mathbf{a}$.

We observe that, in the above two problems, when $\mathbf{z}$ or $\mathbf{a}$ is on or very close to the boundary ∂S, the function value, or its approximation, may be incomputable or very hard to compute. This is because the underlying function cannot, in general, be extended beyond ∂S, and because the problem of determining whether a point $\mathbf{z}$ is on a computable curve Γ is undecidable.

To overcome this problem, we adopt a less restrictive model of computation and allow the oracle Turing machines that compute these functions to make errors when $\mathbf{z}$ or $\mathbf{a}$ is very close to the boundary ∂S. We use a simple rule to control the errors: when the oracle Turing machine computes the value of a function at $\mathbf{z}$ and $\mathbf{a}$ up to precision 2^{-n}, it may make errors if either $\mathbf{z}$ or $\mathbf{a}$ is close to ∂S within a distance of $\leq 2^{-n}$.

Based on this less restrictive model, we are able to characterize the computational complexity of the logarithm problem and the square root problem with the counting complexity classes $\#P$, $\oplus P$ and MP (also denoted $MidBitP$ in literature).Our main results can be stated in terms of the relations between class P and these counting complexity classes:

(1) The logarithm problem is polynomial-time solvable if and only if $FP = \#P$.
(2) If $P = MP$, then the square root problem is polynomial-time solvable.
(3) If $P \neq \oplus P$, then the square root problem is not polynomial-time solvable.

Result (1) reflects the intuition that computing single-valued, analytic branches of the logarithm function is closely related to the computation of winding numbers, which is known to have complexity $P^{\#P}$ [3]. However, compared to the algorithm for the winding number problem in [3], our algorithm for the logarithm and square root problems is more involved and makes use of many properties of simply connected domains and Jordan curves. We include a detailed description of this method in Section 3.

Our basic computational model for real-valued functions and two-dimensional regions is the oracle Turing machine. For the general theory of computable analysis based on the Turing machine model, see, for instance, Pour-El and Richards [12] and Weihrauch [15]. For the theory of computational complexity of real functions based on this computational model, see Ko [9]. The extension of this theory to include the computational complexity of two-dimensional regions has been presented in Chou and Ko [3]. Computational complexity of problems related to two-dimensional regions has been studied recently in several directions [1, 2, 4, 5, 10, 13, 14]. All these works used Turing machine and oracle Turing machine as the basic model.

2 Definitions and Notation

The basic computational objects in continuous computation are dyadic rationals $\mathbb{D} = \{m/2^n : m \in \mathbb{Z}, n \in \mathbb{N}\}$, and we denote $\mathbb{D}_n = \{m/2^n : m \in \mathbb{Z}\}$. We use $\mathbb{R}$ to denote the class of real numbers and $\mathbb{C}$ the class of complex numbers. A complex number $\mathbf{z} = x + iy$ is also denoted by $\langle x, y\rangle$. For any point $\mathbf{z} \in \mathbb{C}$ and any set $S \subseteq \mathbb{C}$, we let $\delta(\mathbf{z}, S)$ be the distance between $\mathbf{z}$ and S; that is, $\delta(\mathbf{z}, S) = \inf\{|\mathbf{z} - \mathbf{z}'| : \mathbf{z}' \in S\}$, where $|\cdot|$ denotes the absolute value.

We say a function $\phi : \mathbb{N} \to \mathbb{D}$ *binary converges* to (or *represents*) a real number x, if (i) for all $n \geq 0$, $\phi(n) \in \mathbb{D}_n$, and (ii) for all $n \geq 0$, $|\phi(n) - x| \leq 2^{-n}$. We say two functions $\phi_x, \phi_y : \mathbb{N} \to \mathbb{D}$ binary converge to (or represent) a complex number $\langle x, y\rangle$ if ϕ_x and ϕ_y binary converge to two real numbers x and y, respectively.

To compute a real-valued function $f : \mathbb{R} \to \mathbb{R}$, we use oracle Turing machines as the computational model. We say an oracle Turing machine M *computes* a function $f : \mathbb{R} \to \mathbb{R}$ if, for a given oracle ϕ that binary converges to a real number x and for a given input $n > 0$, $M^{\phi}(n)$ halts and outputs a dyadic rational e such that $|e - f(x)| \leq 2^{-n}$. We say a function $f : \mathbb{R} \to \mathbb{R}$ is *polynomial-time computable* if there exists a polynomial-time oracle Turing machine that computes f.

The notions of computable and polynomial-time computable real functions can be extended naturally to functions $f : \mathbb{R} \to \mathbb{C}$ and functions $f : \mathbb{C} \to \mathbb{C}$. A Jordan curve (simple, closed curve) Γ in $\mathbb{C}$ is *polynomial-time computable* if there exists a polynomial-time computable function $f : [0, 1] \to \mathbb{C}$ such that the range of f is Γ, f is one-to-one on $[0, 1)$ and $f(0) = f(1)$. It is well known that the interior of a Jordan curve is a simply connected domain.

Let S be a bounded, simply connected domain in $\mathbb{C}$ whose boundary ∂S is a polynomial-time computable Jordan curve. One question is how to define the computability and complexity of a function $g : S \to \mathbb{C}$. Based on the discussions in Section 1, it might be much harder to compute the value $g(\mathbf{z})$ when $\mathbf{z}$ is very close to the boundary ∂S of S, and hence we need to take the distance of $\mathbf{z}$ to the boundary ∂S into consideration. Our approach to resolve this issue is to be less restrictive and allow the machine M that computes the function f to make errors, while the errors are required to be under control. This notion is a generalization of polynomial-time recognizable sets in Chou and Ko [3].

Definition 1 (*a*) *Let S be a bounded, simply connected domain whose boundary ∂S is a computable Jordan curve. A function $f : S \to \mathbb{C}$ is* computable on the domain S *if there exists an oracle Turing machine M such that for any oracles (ϕ, ψ) representing a complex number $\mathbf{z} \in S$, $|M^{\phi,\psi}(n) - f(\mathbf{z})| \leq 2^{-n}$ for all inputs $n > 0$ whenever $\delta(\mathbf{z}, \partial S) > 2^{-n}$.*

(*b*) *Furthermore, f is* polynomial-time computable on the domain S *if f is computable on S by an oracle Turing machine that operates in polynomial time.*

The fundamental complexity classes we are interested in are the class P of sets accepted by deterministic polynomial-time Turing machines, and the class FP of functions (mapping strings to strings) computable by deterministic polynomial-time Turing machines. We will also use in this paper the following complexity classes (see Du and Ko [6] and Green et al [7]):

$\#P$: the class of functions that count the number of accepting paths of nondeterministic polynomial-time machines.

$\oplus P$: the class of sets A such that there exists a nondeterministic polynomial-time Turing machine M such that for all x, $x \in A$ iff there are an odd number of accepting paths for x in M; equivalently, a set A is in $\oplus P$ if there exists a function $G \in \#P$ such that for all x, $x \in A$ if and only if the least significant bit of $G(x)$ is one.

MP_b: the class of sets A for which there exist a function $G \in \#P$ and a function $\phi \in FP$ such that for all x, $x \in A$ if and only if the $\phi(x)$-th bit in the b-ary representation of $G(x)$ is not zero, where b is an integer greater than one.

MP: the union of MP_b over all $b \geq 2$.

3 An Algorithm for Continuous Argument Functions

In this section, we will present a method for computing *continuous argument functions*, which is a critical step for the logarithm problem. Throughout this section, let S be a bounded, simply connected domain whose boundary ∂S is a polynomial-time computable Jordan curve. We assume that ∂S is represented by a polynomial-time computable function $f : [0, 1] \to \mathbb{C}$ and also use f to denote the image of f(i.e., ∂S).

Let $arg(\mathbf{z})$ denote the arguments of $\mathbf{z} \in \mathbb{C}$ if $\mathbf{z} \neq 0$; that is, arg is a multi-valued function from $\mathbb{C} - \{0\}$ to $\mathbb{R}$ such that $\mathbf{z} = |\mathbf{z}|e^{arg(\mathbf{z})i}$ (note that we also

treat $arg(\mathbf{z})$ as a set of real numbers). We define a function $h_S : (S \cup \partial S) \times (\mathbb{C} - (S \cup \partial S)) \to \mathbb{R}$ such that (i) for any fixed point $\mathbf{a} \in (\mathbb{C} - (S \cup \partial S))$, $h_S(\mathbf{z}, \mathbf{a})$ is continuous, (ii) $h_S(f(0), \mathbf{a}) = 0$ for any $\mathbf{a} \in (\mathbb{C} - (S \cup \partial S))$, and (iii) $2\pi \cdot h_S(\mathbf{z}, \mathbf{a})$ equals $\theta_1 - \theta_2$ for some $\theta_1 \in arg(\mathbf{z} - a)$ and some $\theta_2 \in arg(f(0) - \mathbf{a})$. We call this function h_S *the continuous argument function* of S. It is obvious that the imaginary part of $f_1(\mathbf{z} - \mathbf{a}) - f_1(\mathbf{z}_0 - \mathbf{a})$ is $2\pi \cdot h_S(\mathbf{z}, \mathbf{a})$, where f_1 is an arbitrary single-valued, analytic branch of $\log \mathbf{z}$ in the domain $S - \mathbf{a}$ and $\mathbf{z}_0 = f(0)$ is a fixed point on ∂S.

Lemma 2 (*a*) *The logarithm problem on S is polynomial-time solvable if and only if $h_S(\mathbf{z}, \mathbf{a})$ is polynomial-time computable.*

(*b*) *The square root problem on S is polynomial-time solvable if and only if the function $(h_S(\mathbf{z}, \mathbf{a}) \bmod 2)$ is polynomial-time computable.*

Now we consider how to compute $h_S(\mathbf{z}, \mathbf{a})$ for $\mathbf{z} \in \overline{S}$ and $\mathbf{a} \in \mathbb{C} - \overline{S}$. For a point $\mathbf{z} = f(t)$ on ∂S, we can compute $h_S(\mathbf{z}, \mathbf{a})$ by computing the integration of $1/(\mathbf{z} - \mathbf{a})$ over the curve $f([0, t])$, which is a generalization of Theorem 6.4 in Chou and Ko [3]. In fact, the integration method can be applied to any closed curve Γ represented by a polynomial-time computable function f; more precisely, by the integration method, we can compute a *continuous* function $g_f : [0, 1] \times (\mathbb{C} - \Gamma) \to \mathbb{R}$ that satisfies two conditions: (1) $g_f(0, \mathbf{a}) = 0$ for all $\mathbf{a} \in \mathbb{C} - \Gamma$, and (2) $g_f(t, \mathbf{a}) \cdot 2\pi \in \{\theta_1 - \theta_0 : \theta_1 \in arg(f(t), \mathbf{a}), \theta_0 \in arg(f(0), \mathbf{a})\}$ for all $t \in [0, 1]$ and $\mathbf{a} \in \mathbb{C} - \Gamma$. Note that $g_f(1, \mathbf{a})$ is just the winding number of $\mathbf{a}$ with respect to Γ, and $g_f(t, \mathbf{a}) = h_S(f(t), \mathbf{a})$ when f represents the Jordan curve ∂S.

For points $\mathbf{z} \in S$, the situation is more complicated. Intuitively we can compute $h_S(\mathbf{z}, \mathbf{a})$ as follows (see Figure 1):

(a) Let L be the half line starting from $\mathbf{z}$ going in the direction from $\mathbf{a}$ to $\mathbf{z}$. Then, find a real number $t_0 \in [0, 1]$ such that $f(t_0)$ lies on L, and the line segment $\overline{\mathbf{z} f(t_0)}$ lies entirely in $\overline{S}$.
(b) Compute $h_S(f(t_0), \mathbf{a})$ by integration; and let $h_S(\mathbf{z}, \mathbf{a}) = h_S(f(t_0), \mathbf{a}) = g_f(t_0, \mathbf{a})$.

However, we observe that, in general, Step (a) is hard to implement, since, in general, the inverse function $f^{-1}(\mathbf{z})$ might be incomputable (cf. Corollary 4.3 of Ko [9]), or, even if it is computable, might have arbitrarily high complexity (cf. Theorem 4.4 of Ko [9]). Therefore, we cannot follow Step (a) directly. On the other hand, what we really need is just the value of $h_S(f(t_0), \mathbf{a})$ instead of the value of t_0. Our algorithm will explore the curve ∂S to find some candidates t for t_0, and find the correct value of $h_S(f(t_0), \mathbf{a})$ among them.

Our basic idea of computing $h_S(\mathbf{z}, \mathbf{a})$ is as follows. Let L be the half line starting from $\mathbf{z}$ going in the direction from $\mathbf{a}$ to $\mathbf{z}$. We say L and ∂S intersect *non-degenerately* if (1)$L \cap \partial S$ contains only finitely many points, and (2) if $f(t)$ is in $L \cap \partial S$, then ∂S actually crosses L at $f(t)$. It is easy to see that, if L and ∂S intersect non-degenerately, then $L \cap \partial S$ contains an odd number of points $f(t_0), f(t_1), \ldots, f(t_{2m})$ (see Figure 1). Furthermore, the values $g_f(t_i, \mathbf{a})$ can be canceled out in the following sense.

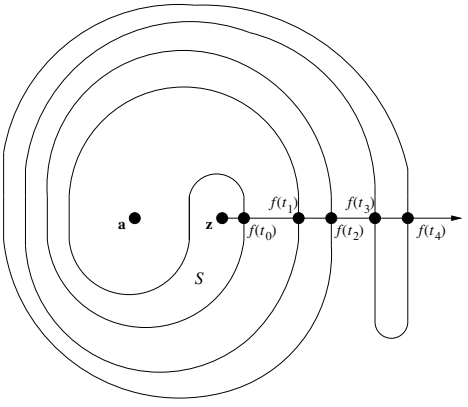


Fig. 1. L and ∂S have an odd number of intersections.

Proposition 3 *Let $f(t_0)$, $f(t_1)$, $\cdots$, $f(t_{2m})$, where $m \geq 0$, be all points in $L \cap \partial S$ in the order of their distances away from $\mathbf{z}$, with $f(t_0)$ being the closest one to $\mathbf{z}$. Then, they satisfy the following properties:*

(1) For any $1 \leq i \leq m$, $\overline{f(t_{2i-1})f(t_{2i})}$ lies entirely in $\overline{S}$, and $g_f(t_{2i-1}, \mathbf{a}) = g_f(t_{2i}, \mathbf{a})$.
(2) For any $1 \leq i \leq m$, f crosses L from opposite directions at points $f(t_{2i-1})$ and $f(t_{2i})$.
(3) $h_S(\mathbf{z}, \mathbf{a}) = g_f(t_0, \mathbf{a})$.

Assume that L and ∂S intersect non-degenerately. Then, we can get values $g_f(t_i, \mathbf{a})$, for all $0 \leq i \leq 2m$, by applying the integration algorithm over the curve to compute $g_f(t, \mathbf{a})$ over all points $f(t)$ at which f crosses L. Note that, in the above computation, we will get all values of $t_0, t_1, \ldots, t_{2m}$, but cannot tell which one is t_0 (because the distance between $f(t_0)$ and $f(t_1)$ could be too small for us to tell which one is closer to $\mathbf{z}$). However, we can still obtain the value of $h_S(\mathbf{z}, \mathbf{a})$ as follows:

(1) Let Δ be an integer such that $g_f(t_i, \mathbf{a}) + \Delta \geq 0$ for all $0 \leq i \leq 2m$.
(2) For each $0 \leq i \leq 2m$, let sgn_i be 1 or -1 according to the direction in which f crosses L at $f(t_i)$.
(3) Let $h_S(\mathbf{z}, \mathbf{a}) = \left| \sum_{i=0}^{2m} (sgn_i \cdot (g_f(t_i, \mathbf{a}) + \Delta)) \right| - \Delta$.

That is, we use the factor sgn_i to cancel out the values of $g_f(t_{2i-1}, \mathbf{a}) + \Delta$ and $g_f(t_{2i}, \mathbf{a}) + \Delta$ in the summation of (3), and the only one left is $g_f(t_0, \mathbf{a}) + \Delta$. Yet we do not know what the value t_0 is. We use the extra term Δ so that $g_f(t_0, \mathbf{a})$ can be extracted from the absolute value of $sgn_0 \cdot (g_f(t_0, \mathbf{a}) + \Delta)$.

There are some technical problems with these ideas. First, since we can only approximate ∂S, we may not be able to compute the intersections of L and ∂S correctly. Second, L and ∂S may not intersect non-degenerately. That is, one of the following situations may occur:

(1) $L \cap \partial S$ contains infinitely many points (e.g., the curve ∂S may cross L infinitely many times); or
(2) $f(t)$ is in $L \cap \partial S$ for some $t \in [0,1]$, but f does not cross L at $f(t)$.

In the following, we describe a method to approximate intersections of ∂S and L that will solve the above problems. The main idea is to use a piecewise linear curve f_n that approximates f and apply the integration method on f_n. By a careful analysis, we can show that this computation is still correct.

First, without loss of generality, we assume that $\mathbf{z}$ and $\mathbf{a}$ are dyadic points. (Thus, we can tell whether a dyadic point lies on L or not.) Next, let M be an oracle Turing machine that computes f in time p for some polynomial p. It follows that p is a modulus function of f. Let n be an integer such that $\delta(\mathbf{z}, \partial S) > 2^{-n}$ and $\delta(\mathbf{a}, \partial S) > 2^{-n}$. For each $0 \le i \le 2^{p(2n)}$, let $s_i = i \cdot 2^{-p(2n)}$ and $\mathbf{z}_i = M^{b_{s_i}}(2n)$, where b_x is the *standard Cauchy function* of x such that $x - 2^{-n} < b_x(n) \le x$ for all n. Then, for any $0 \le i \le 2^{p(2n)} - 1$ and any $t \in [s_i, s_{i+1}]$, we have $|f(t) - \mathbf{z}_i| \le 2^{-2n}$. Let f_n be the piecewise linear function with breakpoints $f_n(s_i) = \mathbf{z}_i$, for $i = 0, \ldots, 2^{p(2n)}$, and Γ_n be the image of f_n on $[0,1]$. Then, Γ_n is an approximation of ∂S within an error 2^{-2n}. Note that Γ_n is not necessarily simple.

We now define a function sgn_{f_n} which assigns value $+1$, -1 or 0 to each directed line segment of Γ_n according to whether it crosses L counter-clockwisely, or crosses L clockwisely, or does not cross L, respectively[1]. When $sgn_{f_n}(i)$ is not zero, there is a unique intersection $\mathbf{z}'_i := f_n(s'_i)$ of $f_n([s_{i-1}, s_i])$ and L.

Lemma 4 *Assume that ∂S is polynomial-time computable. Then, the functions $\phi_1(n,i) = sgn_{f_n}(i)$, $\phi_2(n,i) = s'_i$ and $\phi_3(n,i) = \mathbf{z}'_i$ (if they exist) are polynomial-time computable.*

For simplicity, we may assume that $f_n(0) = f(0)$ (this can be achieved by, for example, transforming the whole plane so that $f(0)$ becomes the origin).

Theorem 5 *Assume hereafter $s'_i = s_i$ if $sgn_{f_n}(i) = 0$. We have* $\left| h_s(\mathbf{z}, \mathbf{a}) \right| = \left| \sum_{i=1}^{2^{p(2n)}} sgn_{f_n}(i) \cdot g_{f_n}(s'_i, \mathbf{a}) \right|$.

To prove Theorem 5, we need to show that

(1) In the right hand of the formula in Theorem 5, all values but one of $g_{f_n}(s'_i, \mathbf{a})$ are canceled out. (Since the curve Γ_n is only an approximation to Γ, it is not necessarily a simple curve and hence this fact does not follow from Proposition 3 immediately.)
(2) The remaining term is $\pm h_S(\mathbf{z}, \mathbf{a})$.

[1] Note that the line that contains L divides the plane into two half planes. We arbitrarily include L in one of the half planes. By a line segment Λ crossing L, we mean that $\Lambda \cap L \neq \emptyset$ and the two endpoints of Λ are in two different half planes.

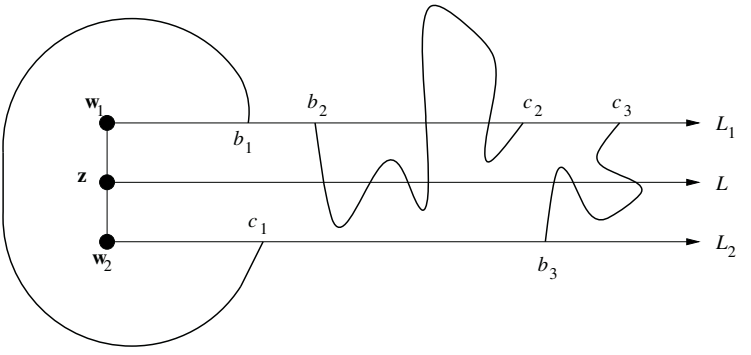


Fig. 2. Curve Γ on three different types of intervals.

To do this, we divide the interval $[0,1]$ into a finite number of subintervals, and examine each subinterval separately. First, define $\mathbf{w}_1 \in S_1$, $\mathbf{w}_2 \in S_2$ to be the two points which have distance $2^{-(2n-1)}$ from $\mathbf{z}$ and the line segment $\overline{\mathbf{w}_1\mathbf{w}_2}$ is perpendicular to L (see Figure 2). For each $i = 1, 2$, define a half line L_i that starts at $\mathbf{w}_i$ and runs parallel to L. Call the domain between the two half lines L_1 and L_2 and the line segment $\overline{\mathbf{w}_1\mathbf{w}_2}$ (including the boundary) the *crossing zone*. We let Z denote the crossing zone.

Without loss of generality, assume that $\mathbf{z}_0 = f(0)$ is not in the crossing zone. We call an interval $[b,c] \subseteq [0,1]$ a *crossing interval* if $[b,c]$ is a maximal interval with the following properties: (i) $f(b) \in L_i$ and $f(c) \in L_{3-i}$ for $i = 1$ or 2, and (ii) $f([b,c])$ lies entirely in the crossing zone Z. (By "maximal" we mean that all intervals $[b',c']$ that properly contain $[b,c]$ do not satisfy both (i) and (ii).) We call an interval $[a,b]$ a *non-crossing interval* if (i) $f(b) \in L_i$ and $f(c) \in L_{3-i}$ for $i = 1$ or 2 (or if $b = 0$ and $f(c) \in L_1 \cup L_2$, or $c = 1$ and $f(b) \in L_1 \cup L_2$), and (ii) $f((b,c))$ lies entirely outside the crossing zone Z. If we remove all crossing intervals and non-crossing intervals from $[0,1]$, the remainder is the union of finite intervals. We call each such interval a *semi-crossing interval*. A semi-crossing interval $[b,c]$ satisfies the following conditions: (i) both $f(a)$ and $f(b)$ are in L_i for $i = 1$ or 2, and (ii) if $f(a) \in L_i$ for $i = 1$ or 2, then $f([a,b]) \cap L_{3-i} = \emptyset$. Figure 2 shows the curve Γ on these intervals, where $[b_1, c_1]$ is a non-crossing interval, $[b_2, c_2]$ is a semi-crossing interval, and $[b_3, c_3]$ is a crossing interval.

Lemma 6 *Let $\mathbf{z}_1, \mathbf{z}_2$ be two points in $\overline{S} \cap L$. Assume that there is a path π from $\mathbf{z}_1$ to $\mathbf{z}_2$ that lies in $\overline{S} \cap Z$. Then, $h_S(\mathbf{z}_1, \mathbf{a}) = h_S(\mathbf{z}_2, \mathbf{a})$.*

Lemma 7 *Let $[b,c]$ be a non-crossing interval. Then, $sgn_{f_n}(i) = 0$ for all $s_i \in [b,c]$, and so $\sum_{s_i \in [b,c]} sgn_{f_n}(i) \cdot g_{f_n}(s_i', \mathbf{a}) = 0$.*

Lemma 8 *Let $[b,c]$ be a semi-crossing interval. Then, there are an even number of intersection points $f_n(r_1), f_n(r_2), \cdots, f_n(r_{2m})$ in $f_n([b,c]) \cap L$, and g_{f_n} has the same value $g_{f_n}(r_1, \mathbf{a})$ at all r_i's. These values all cancel out after considering the crossing directions; that is, $\sum_{s_i \in [b,c]} sgn_{f_n}(i) \cdot g_{f_n}(s_i', \mathbf{a}) = 0$.*

For each crossing interval $[b, c]$, define $sgn_{[b,c]} = +1$ if $f(b) \in L_2$ and $f(c) \in L_1$; and $sgn_{[b,c]} = -1$ otherwise.

Lemma 9 *Let $[b, c]$ be a crossing interval.*

(1) There exists at least one point $t \in [b, c]$ such that $f(t) \in L$. For any two such numbers $t_1, t_2 \in [b, c]$ with $f(t_1), f(t_2) \in L$, $g_f(t_1, \mathbf{a}) = g_f(t_2, \mathbf{a})$.

(2) There are an odd number of intersection points $f_n(r_0), f_n(r_1), \ldots, f_n(r_{2m})$ in $f_n([b, c]) \cap L$, and they all have the same value $g_{f_n}(r_i, \mathbf{a})$ as $g_f(t, \mathbf{a})$. All but one of these values cancel out after considering the crossing directions; that is, $\sum_{s_i \in [b,c]} sgn_{f_n}(i) \cdot g_{f_n}(s_i, \mathbf{a}) = sgn_{[b,c]} g_f(t, \mathbf{a})$.

Theorem 5 follows Lemmas 6-9. Based on Theorem 5, we can design an algorithm to compute $h_S(\mathbf{z}, \mathbf{a})$, which, due to the space limit, is omitted.

Theorem 10 *Let S be a bounded, simply connected domain whose boundary ∂S is a polynomial-time computable Jordan curve. Then there exists an oracle Turing machine that computes $h_S(\mathbf{z}, \mathbf{a})$ in polynomial time using a function G in $\#P$ as an oracle.*

Corollary 11 *If $FP = \#P$, then the continuous argument function problem and the logarithm problem are polynomial-time solvable.*

When we apply this algorithm to the square root problem, we only need a constant number of bits from the oracle function G of Theorem 10. Therefore,we get the following corollary.

Corollary 12 *If $P = MP$, then the square root problem is polynomial-time solvable.*

4 Lower Bounds for the Logarithm and Square Root Problems

We have shown, in the last section, that $P^{\#P}$ is an upper bound for the complexity of computing the continuous argument functions, and hence the logarithm problem. On the other hand, we can obtain a lower bound of $P^{\#P}$ with a construction similar to that for the lower bound of the winding number problem of Chou and Ko [3].

Theorem 13 *If $FP \neq \#P$, then there exists a bounded, simply connected domain S whose boundary ∂S is a polynomial-time computable Jordan curve, such that the logarithm problem on S is not polynomial-time computable.*

For the square root problem, we are only able to show a lower bound of $P^{\oplus P}$, which is weaker than the upper bound P^{MP} of Corollary 12.

Theorem 14 *If $P \neq \oplus P$, then there exists a bounded, simply connected domain S whose boundary ∂S is a polynomial-time computable Jordan curve such that the square root problem on S is not polynomial-time solvable.*

References

1. M. Braverman. Hyperbolic Julia sets are poly-time computable. In *Proceedings of the 6th Workshop on Computability and Complexity in Analysis '2004*, *Electronic Notes in Theoretical Computer Science*, 120:17–30, Elsevier, Amsterdam, 2005.
2. M. Braverman and M. Yampolsky. Non-computable julia sets. *CoRR*, math.DS/0406416, 2004.
3. A. W. Chou and K.-I. Ko. Computational complexity of two-dimensional regions. In *SIAM.J.COMPUT.*, 24:923–947, October 1995.
4. A. W. Chou and K.-I. Ko. On the complexity of finding paths in a two-dimensional domain I: Shortest paths. *Mathematical Logic Quarterly*, 50(6):551–572, 2004.
5. A. W. Chou and K.-I. Ko. On the complexity of finding paths in a two-dimensional domain II: Picewise straight-line paths. *Proceedings of the 6th Workshop on Computability and Complexity in Analysis '2004*, *Electronic Notes in Theoretical Computer Science*, 120:45–57, Elsevier, Amsterdam, 2005.
6. D.-Z. Du and K.-I. Ko. *Theory of Computational Complexity*. John Wiley & Sons, New York, 2000.
7. F. Green, J. Köbler, K. Regan, T. Schwentick, and J. Torán. The power of the middle bit of a #P function. *Journal of Computer and System Sciences*, 50:456–467, 1995.
8. P. Herici. *Applied and Computational Complex Analysis*, volumes 1-3. John Wiley & Sons, New York, 1974.
9. K.-I. Ko. *Complexity Theory of Real Functions*. Birkhäuser, Boston, 1991.
10. K.-I. Ko. Computational complexity of fractals. *Proceedings of the 7th and 8th Asian Logic Conferences*, 252–269, World Scientific, Singapore, 2003.
11. K.-I. Ko and H. Friedman. Computational Complexity of Real Functions. *Theoretic Computer Science*, 20:323–352, 1982.
12. M. B. Pour-El and J. I. Richards. *Computability in Analysis and Physics*. Perspectives in Mathematical Logic, Springer, Berlin, 1989.
13. R. Rettinger. A fast algorithm for Julia sets of hyperbolic rational functions. *Proceedings of the 6th Workshop on Computability and Complexity in Analysis '2004*, *Electronic Notes in Theoretical Computer Science*, 120:145–157, Elsevier, Amsterdam, 2005.
14. R. Rettinger and K. Weihrauch. The computational complexity of some julia sets. *the Thirty-Fifth Annual ACM Symposium on Theory of Computing (STOC)*, 177–185, 2003.
15. K. Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

Solovay Reducibility on D-c.e Real Numbers*

Robert Rettinger[1] and Xizhong Zheng[2,3,**]

[1] Theoretische Informatik II, FernUniversität Hagen, D-58084 Hagen, Germany
[2] Department of Computer Science, Jiangsu University, Zhenjiang 212013, China
[3] Theoretische Informatik, BTU Cottbus, D-03044 Cottbus, Germany
zheng@informatik.tu-cottbus.de

Abstract. A c.e. real x is Solovay reducible to another c.e. real y if x can be approximated at least as efficiently as y by means of increasing computable sequences of rational numbers. The Solovay reducibility classifies elegantly the relative randomness of c.e. reals. Especially, the c.e. random reals are complete unter the Solovay reducibility for c.e. reals. In this paper we investigate an extension of the Solovay reducibility to the Δ_2^0-reals and show that the c.e. random reals are complete under (extended) Solovay reducibility for d-c.e. reals too. Actually we show that only the d-c.e. reals can be Solovay reducible to an c.e. random real. Furthermore, we show that this fails for the class of divergence bounded computable reals which extends the class of d-c.e. reals properly. In addition, we show also that any d-c.e. random reals are either c.e. or co-c.e.

1 Introduction

Randomness is a very important property in mathematics and computer science beside the computability. While a computable object has a very simple structure and can be effectively characterized, a random object should have the maximal irregularity. One of the most popular description of randomness is the definition of Martin-Löf [10]: an infinite binary sequence α is called *(Martin-Löf) random* if α passes all Martin-Löf tests $\{U_n : n \in \mathbb{N}\}$, i.e., $\alpha \notin \bigcap_{n\in\mathbb{N}} U_n$. Here a Martin-Löf test is a computable collection $\{U_n : n \in \mathbb{N}\}$ of computably enumerable open sets of the Cantor-space $\{0,1\}^\omega$ such that $\mu(U_n) \leq 2^{-n}$. The class $\bigcap_{n\in\mathbb{N}} U_n$ has Lebesgue measure zero if $\{U_n : n \in \mathbb{N}\}$ is a Martin-Löf test and it is a "small class". Thus, a random sequence is "typical" because it does not belong to such kind of small (and hence special) groups. In other words, the Martin-Löf randomness reflects the *typicalness* of random objects.

Another characterization of random objects is their "incompressibility" or "chaoticness". That is, a random object should have the most complex structure. Thus, a sequence is random if there is no better way to describe it than to write it down literally. More precisely, let a *prefix machine* be a Turing machine with a prefix-free domain and define the *prefix-free complexity* of a binary string σ as $K(\sigma) := \min\{|\tau| : U(\tau) = \sigma\}$, where U is a universal prefix-free machine.

* This work is supported by DFG (446 CHV 113/240/0-1) and NSFC (10420130638).
** Corresponding author

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 359–368, 2005.

A binary sequence α is called *Kolmogorov-Levin-Chaitin random*, if there is a constant c such that $K(\alpha \upharpoonright n) \geq n - c$ for all n (see, e.g., [3, 9]). Thus, random sequences have the highest prefix-free complexity.

The Martin-Löf randomness and Kolmogorov-Levin-Chaitin randomness are equivalent (see [3]). However, the latter leads naturally to the investigation of relative randomness as follows. For any binary sequences α and β, if there is a constant c such that $K(\alpha \upharpoonright n) \leq K(\beta \upharpoonright n) + c$ for all n, then β is at least as random as α and we say that α is *K-reducible* to β (denoted by $\alpha \leq_K \beta$). In general, if we consider a reducibility $\leq$ as a measure of relative randomness, then it should at least have the following *Solovay property*:

$$\alpha \leq \beta \Longrightarrow (\exists c)(\forall n)(K(\alpha \upharpoonright n) \leq K(\beta \upharpoonright n) + c). \tag{1}$$

For the real numbers, their randomness can be regarded as the randomness of their binary expansions. Several reducibilities on real numbers with Solovay property have been proposed ([6, 12, 14]). Especially, for the c.e. reals (the limits of increasing computable sequences of rational numbers) there is a very elegant reducibility – the Solovay reducibility – defined as follows ([12]): a c.e. real x is *Solovay reducible* to another c.e. real y (denoted by $x \leq_S y$) if there are increasing computable sequences (x_s) and (y_s) of rational numbers which converges to x and y, respectively, such that

$$(\exists c)(\forall n)(x - x_n \leq c \cdot (y - y_n)). \tag{2}$$

That is, x can be approximated at least as efficiently as y by increasing computable sequences of rational numbers. Obviously, $\leq_S$ is reflexive and transitive and the deduced equivalent classes are called *Solovay degrees* or simply S-degrees. Solovay [12] shows that $\leq_S$ has the Solovay property and hence classifies c.e. reals into different levels according to their relative randomness. Downey, Hirschfeldt and Nies [7] have shown that the S-degrees of c.e. reals form a dense distributive uppersemilattice.

One of the most interesting results about Solovay reducibility are the equivalent descriptions of Solovay complete c.e. reals. A c.e. real x is called *Solovay complete* or *S-complete*[1] for c.e. reals if it is complete under Solovay reducibility, i.e., $y \leq_S x$ for all c.e. reals y. The S-complete reals relate very closely to the c.e. random reals and Ω-numbers. First, Chaitin [3] called the halting probability x of a universal prefix machine U an *Ω-number*, i.e., $x = \Omega_U := \sum_{U(\sigma)\downarrow} 2^{-|\sigma|}$ and showed that Ω-numbers are c.e. and random. Then, Solovay [12] observed that any Ω-number is S-complete. The other direction, i.e., S-complete reals are just the Ω-numbers, were proven by Calude, Hertling, Khoussainov and Wang [2]. Finally, Kuçera and Slaman [8] closed this circle by showing that any c.e. random real is S-complete. That is, we have the following result.

Theorem 1.1 (Chaitin, Solovay, Kuçera, Slaman and Calude et al). *A c.e. real x is random iff it is S-complete iff it is an Ω-number.*

[1] Originally Solovay [12] calls it *Ω-like*.

In order to extend the Solovay reducibility to larger classes of reals Downey, Hirschfeldt and LaForte [6] have introduced two new reducibilities, $\leq_{sw}$ (strongly weak truth-table reducibility) and $\leq_{rH}$ (relative H reducibility). Both reducibilities are defined on all reals and have the Solovay property and some other nice properties. However, none of them coincide with the Solovay reducibility on c.e. reals and hence are not really extensions of $\leq_S$. In [14], the authors suggested another reducibility, the extended Solovay reducibility, by considering general instead of increasing computable sequences of rational numbers and replacing condition (2) by the following weaker condition

$$(\exists c)(\forall n)(|x - x_n| \leq c \cdot (|y - y_n| + 2^{-n})). \tag{3}$$

Thus, the extended Solovay reducibility is defined on all *computably approximable* reals (c.a. reals for short, i.e., the limits of computable sequences of rational numbers, see [1]). Since the extended and original Solovay reducibility coincide on c.e. reals, it is simply called Solovay reducibility and denoted by $\leq_S$ too. In this paper we investigate mainly the properties of S-reducibility on d-c.e. reals (the differences of c.e. reals) and show that a similar result like Theorem 1.1 holds for the class of d-c.e. reals too. That is, a d-c.e. real is Solovay complete for the class of d-c.e. reals if and only if it is c.e. random if and only if it is an Ω-number. For a larger class of divergence bounded computable reals however, this fails.

2 D-c.e. Real Numbers

We begin with the discussion of the class of d-c.e. reals in this section. By definition, a real x is *d-c.e.* (difference of c.e.) if there are c.e. reals y and z such that $x = y - z$. The class of all d-c.e. reals is denoted by **DCE**. This class has another very interesting equivalent characterization.

Theorem 2.1 (Ambos-Spies, Weihrauch and Zheng [1]). *A real x is d-c.e. iff there is a computable sequence (x_s) of rational numbers which converges to x* weakly effectively *in the sense that $\sum_{s\in\mathbb{N}} |x_s - x_{s+1}| \leq c$ for a constant c.*

Since the computable reals are limits of computable sequences of rational numbers which converge *effectively* in the sense that $|x_s - x_{s+1}| \leq 2^{-s}$ for all s, d-c.e. reals are also called *weakly computable* because of Theorem 2.1. The class of weakly computable reals is denoted also by **WC**. In addition, by means of Theorem 2.1 it is easy to show that the class of d-c.e. reals is closed under arithmetical operations $+, -, \times$ and $\div$ and hence it is a field[2]. Thus, the class **WC** is the arithmetical closure of c.e. reals.

While the effective convergence condition $|x_s - x_{s+1}| \leq 2^{-s}$ can be replaced by $|x - x_s| \leq 2^{-s}$ and we still get the same class of computable reals, the weak convergence condition $\sum_{s\in\mathbb{N}} |x_s - x_{s+1}| \leq c$ cannot be equivalently replaced by, say, $\sum_{s\in\mathbb{N}} |x - x_s| \leq c$ for a constant c. In fact, if x is the limit of a computable

[2] It is actually a real closed field as shown recently by Raichev [11].

sequence (x_s) of rational numbers such that $\sum_{s\in\mathbb{N}} |x - x_s| \leq c$ for some constant c, then x is computable! Besides, the condition $|x - x_s| \leq c_s$ for a computable sequence (c_s) converging to 0 implies also the computability of x. However, if we consider a computable sequence (c_s) of c.e. reals converging to 0, we get another equivalent characterization of the class of d-c.e. reals.

Definition 2.2. *A sequence (x_s) of reals converges to x* computably enumerably bounded *(c.e. bounded) if there is a computable sequence (δ_s) of positive rational numbers such that $\sum_{s\in\mathbb{N}} \delta_s$ is finite and*

$$|x - x_s| \leq \sum_{i\geq s} \delta_i \tag{4}$$

for all $s \in \mathbb{N}$.

Notice that, if a sequence (x_s) converges effectively to x, then $|x - x_s| \leq 2^{-s}$ hold for all s. Therefore, it is natural to say that the sequence (x_s) converges *computably bounded* if (x_s) converges effectively. For a c.e. bounded convergent sequence (x_s), a to zero convergent computable bound for $|x - x_s|$ is not available in general. The bounds $\sum_{i\geq s} \delta_i$ in (4) are only c.e. reals and they converge to 0 monotonically when s increases. The c.e. bounded convergence supplies another characterization of the class of d-c.e. reals as follows.

Theorem 2.3. *A real number x is d-c.e. if and only if there is a computable sequence (x_s) of rational numbers converging to x c.e. bounded.*

Proof. Suppose that x is d-c.e., i.e., $x = y - z$ for c.e. reals y and z. Let (y_s) and (z_s) be increasing computable sequences of rational numbers which converge to y and z, respectively. Define $x_s := y_s - z_s$ for all s. Then (x_s) is a computable sequence of rational numbers which satisfies

$$\begin{aligned} |x - x_s| &= |(y - z) - (y_s - z_s)| \leq (y - y_s) + (z - z_s) \\ &= \sum\nolimits_{i\geq s} ((y_{s+1} - y_s) + (z_{s+1} - z_s)) \end{aligned}$$

for all s. Thus, the sequence (x_s) converges c.e. bounded with respect to the sequence (δ_s) defined by $\delta_s := (y_{s+1} - y_s) + (z_{s+1} - z_s)$ for all s.

On the other hand, let (x_s) and (δ_s) (for $\delta_s > 0$) be computable sequences of rational numbers which satisfy condition (4). We show that x is d-c.e. By Theorem 2.1, it suffices to construct a computable sequence (x'_s) of rational numbers which converges to x weakly effectively. The sequence (x'_s) is defined inductively by $x'_0 := x_0$ and

$$x'_{s+1} := \begin{cases} x_{s+1} & \text{if } |x'_s - x_{s+1}| \leq \delta_{s+1}; \\ x'_s + \delta_{s+1} & \text{if } |x'_s - x_{s+1}| > \delta_{s+1} \text{ and } x'_s < x_{s+1}; \\ x'_s - \delta_{s+1} & \text{if } |x'_s - x_{s+1}| > \delta_{s+1} \text{ and } x'_s > x_{s+1}, \end{cases} \tag{5}$$

for all s. Obviously, (x'_s) is a computable sequence of rational numbers. Now we prove by induction on s that the sequence (x'_s) satisfies the following condition:

$$(\forall s \in \mathbb{N}) \left(|x'_s - x'_{s+1}| \leq \delta_{s+1} \;\&\; |x - x'_{s+1}| \leq \sum\nolimits_{i>s} \delta_i \right). \tag{6}$$

Assume by induction hypothesis that (6) holds for all $s < n$. For $s = n$, the inequality $|x'_n - x'_{n+1}| \leq \delta_{n+1}$ follows directly from definition (5). For the second inequality of (6), we consider the intervals

$$I_t := \left[x - \sum\nolimits_{i \geq t} \delta_i;\ x + \sum\nolimits_{i \geq t} \delta_i\right]$$

for $t := n$ and $t := n + 1$. Then condition (4) says that $x_t \in I_t$ for all t. By induction hypothesis for $s = n - 1$, we have $|x - x'_n| \leq \sum_{i>n-1} \delta_i = \sum_{i \geq n} \delta_i$. That is, $x'_n \in I_n$. It suffices now to show that $x'_{n+1} \in I_{n+1}$.

If $x'_n \in I_{n+1}$, i.e., both x'_n and x_{n+1} belong the interval I_{n+1}, then the definition (5) implies that $x'_{n+1} \in I_{n+1}$. Suppose now that $x'_s \in I_n \setminus I_{n+1}$. If $|x'_n - x_{n+1}| \leq \delta_{n+1}$, then we have $x'_{n+1} = x_{n+1} \in I_{n+1}$. Otherwise, if $|x'_n - x_{n+1}| > \delta_{n+1}$ and $x'_n < x_{n+1}$, then we have $x'_{n+1} = x'_n + \delta_{n+1} \in I_{n+1}$ too, because the distance between the left endpoints of the intervals I_n and I_{n+1} is just δ_{n+1}. Similarly, we have $x'_{n+1} \in I_{n+1}$ too, if $x'_n > x_{n+1}$. This implies that $|x - x'_{n+1}| \leq \sum_{i>n} \delta_i$, and hence (6) holds for all s.

The second inequality of (6) implies that the sequence (x'_s) converges to x. The first inequality of (6) implies furthermore that this convergence is weakly effective, i.e. $\sum_{s \in \mathbb{N}} |x'_s - x'_{s+1}| \leq \sum_{s \in \mathbb{N}} \delta_s$. Thus, x is a d-c.e. real by the Theorem 2.1.

As shown in [1], the class **WC** is the arithmetical closure of the class **CE** of c.e. reals and $\mathbf{CE} \subsetneq \mathbf{WC}$. However we will show that any d-c.e. random real is either c.e. or co-c.e, where co-c.e. reals are limits of decreasing computable sequences of rational numbers. To this end, we need another characterization of random reals by Solovay. We call a set $U \subseteq \mathbb{R}$ of reals *c.e. open* if there is a computable sequence (a_s, b_s) of rational open intervals such that $U = \bigcup_{s \in \mathbb{N}} (a_s, b_s)$.

Theorem 2.4 (Solovay [12]). *A real x is Martin-Löf random iff x is in only finitely many U_i for any computable collection $\{U_n : n \in \mathbb{N}\}$ of c.e. open sets such that $\sum_{n \in \mathbb{N}} \mu(U_n) < \infty$, where $\mu(U)$ is the Lebesgue measure of U.*

Theorem 2.5. *Any random d-c.e. real is either c.e. or co-c.e.*

Proof. Let x be a d-c.e. random real. By Theorem 2.1, there is a computable sequence (x_s) of rational numbers which converges to x weakly effectively, i.e., $\sum_{s \in \mathbb{N}} |x_{s+1} - x_s| < c$ for a constant c.

Assume by contradiction that x is neither c.e. nor co-c.e. Then there are infinitely many s such that $x_s < x$. Otherwise, x is co-c.e. if $x_s \geq x$ for almost all s. Analogously, there are infinitely many s such that $x_s > x$ too. Thus, if $x_s < x$ for some $s \in \mathbb{N}$, then there is a $t \geq s$ such that $x_t < x < x_{t+1}$ and if $x_s > x$ then there is a $t \geq s$ such that $x_t > x > x_{t+1}$. This implies further that there are infinitely many s such that $x_s < x < x_{s+1}$. Let $U_s := (x_s, x_{s+1})$ if $x_s < x_{s+1}$ and $U_s := \emptyset$ otherwise. Then $\{U_n : n \in \mathbb{N}\}$ is a computable collection of c.e. open sets such that $\sum_{n \in \mathbb{N}} \mu(U_n) = \sum_{s \in \mathbb{N}} (x_{s+1} \dot{-} x_s) \leq \sum_{s \in \mathbb{N}} |x_{s+1} - x_s| \leq c$, where $x \dot{-} y := x - y$ if $x \geq y$ and $x \dot{-} y := 0$ otherwise. Since x belongs to infinitely many U_n, x is not random by Theorem 2.4. This contradicts our assumption.

Corollary 2.6. *A d-c.e. real is random if and only if it is an Ω-number.*

3 Solovay Completeness for D-c.e. Reals

The original Solovay reducibility is defined only on the c.e. reals by comparing approximation rates of c.e. reals by increasing computable sequences of rational numbers. Its straightforward extension to computably approximable reals is not a proper reducibility, because it is not even transitive (see [14]). However, by a minimal modification, the Solovay reducibility can be reasonably extended to c.a. reals as follows.

Definition 3.1. *A c.a. real x is* Solovay reducible *to another c.a. real y (denoted by $x \leq_S y$) if there are two computable sequences (x_s) and (y_s) of rational numbers which converge to x and y, respectively, and a constant c such that*

$$(\forall s)\left(|x - x_s| \leq c(|y - y_s| + 2^{-s})\right). \tag{7}$$

As shown in [14], this extended Solovay reducibility is reflexive, transitive and hence is a reasonable reducibility. It has the Solovay property (1) and coincides with the original Solovay reducibility on c.e. reals. It is not very difficult to see that the (extended) Solovay reducibility can be defined equivalently in another way described in the next lemma.

Lemma 3.2. *A real x is Solovay reducible to y iff for any computable sequence (y_s) of rational numbers converging to y, there exist a computable sequence (x_s) of rational numbers converging to x and a constant c which satisfy condition (7).*

The computable sequences of rational numbers used in the Definition 3.1 can also be equivalently replaced by computable sequences of reals as shown in the next lemma. Here a sequence (x_s) of real numbers is computable if there is a computable double sequence (r_{st}) of rational numbers such that $|x_s - r_{st}| \leq 2^{-t}$ for all s and t. Obviously, any computable sequence of rational numbers is a computable sequence of real numbers too.

Lemma 3.3. *A real x is Solovay reducible to y iff there are two computable sequences (x_s) and (y_s) of real numbers which converge to x and y, respectively, and a constant c which satisfy condition (7).*

Proof. Let (x_s) and (y_s) be computable sequences of real numbers converging to x and y, respectively, and satisfy condition (7) for a constant c. By definition there are computable double sequences (u_{st}) and (v_{st}) of rational numbers such that $|x_s - u_{st}| \leq 2^{-t}$ and $|y_s - v_{st}| \leq 2^{-t}$ for all $s, t \in \mathbb{N}$. Let $a_s := u_{ss}$ and $b_s := v_{ss}$ for all s. Then (a_s) and (b_s) are computable sequences of rational numbers which converge to x and y, respectively such that

$$\begin{aligned} |x - a_s| &\leq |x - x_s| + |x_s - a_s| \leq c(|y - y_s| + 2^{-s}) + 2^{-s} \\ &\leq c|y - b_s| + c|b_s - y_s| + (c+1)2^{-s} \leq (2c+1)(|y - b_s| + 2^{-s}) \end{aligned}$$

for all $s \in \mathbb{N}$. According to Definition 3.1, x is Solovay reducible to y.

Now we are going to show that, the Ω-numbers are Solovay complete for the class **WC**. Since **WC** is the smallest field containing all c.e. reals, it suffices to show that the class of reals which are Solovay reducible to an Ω-number is a field too. For any real number x, let $S(\leq x)$ denote the class of all reals which are Solovay reducible to x, i.e., $S(\leq x) := \{y \in \mathbb{R} : y \leq_S x\}$. We show first a general result that $S(\leq x)$ is a field for any c.a. real x. To this end, we prove that it is closed unter some class of computable real functions.

Definition 3.4. *A real function $f : \mathbb{R}^n \to \mathbb{R}$ is called* locally Lipschitz *if for each $\boldsymbol{x} \in \mathrm{dom}(f)$ there is a neighborhood U of $\boldsymbol{x}$ and a Lipschitz-constant L such that*

$$(\forall \boldsymbol{u}, \boldsymbol{v} \in U)(|f(\boldsymbol{u}) - f(\boldsymbol{v})| \leq L \cdot |\boldsymbol{u} - \boldsymbol{v}|), \tag{8}$$

where $|\boldsymbol{u} - \boldsymbol{v}| := \sum_{i=1}^{n} |u_i - v_i|$ for $\boldsymbol{u} = (u_1, \ldots, u_n)$ and $\boldsymbol{v} = (v_1, \ldots, v_n)$.

Theorem 3.5. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a locally Lipschitz computable function and let d be a computably approximable real number. The class $S(\leq d)$ is closed under the function f.*

Proof. We prove only the case for $n = 2$. The proofs for other n are similar.

Let d be a c.a. real and let (d_s) be a computable sequence of rational numbers which converges to d. For any reals x, y such that $x \leq_S d$ and $y \leq_S d$, by Lemma 3.2, there are computable sequences (x_s) and (y_s) of rational numbers which converges to x and y, respectively, and a common constant c such that

$$|x - x_s| \leq c(|d - d_s| + 2^{-s}) \quad \text{and} \quad |y - y_s| \leq c(|d - d_s| + 2^{-s})$$

for all $s \in \mathbb{N}$.

Let L be a Lipschitz-constant which satisfies condition (8). By the sequential computability of the computable function f, the sequence (z_s) defined by $z_s := f(x_s, y_s)$ for all s is a computable computable sequence of real numbers. This sequence satisfies furthermore the following condition.

$$\begin{aligned} |f(x, y) - z_s| = |f(x, y) - f(x_s, y_s)| &\leq L(|x - x_s| + |y - y_s|) \\ &\leq 2cL(|d - d_s| + 2^{-s}) \end{aligned}$$

for all $s \in \mathbb{N}$. By Lemma 3.3 we have $f(x, y) \leq_S d$ and hence $f(x, y) \in S(\leq d)$. That is, the class $S(\leq d)$ is closed under the function f.

As an immediate corollary we have

Corollary 3.6. *The class $S(\leq y)$ is a field for any c.a. real y.*

Proof. The functions addition, subtraction, multiplication and division are obviously locally Lipschitz computable functions. Therefore, the class $S(\leq y)$ is closed under arithmetical operations and is a field.

Now we can prove our main result.

Theorem 3.7. *If y is an Ω-number, then $S(\leq y) = \mathbf{WC}$.*

Proof. Let y be an Ω-number. By Theorem 1.1, y is Solovay complete for the class of c.e. reals. That is, the class $S(\leq y)$ contains all c.e. reals. Because $S(\leq y)$ is a field (Corollary 3.6) and **WC** is the smallest field containing all c.e. reals, we have $\mathbf{WC} \subseteq S(\leq y)$.

On the other hand, if $x \in S(\leq y)$, we show that x is d-c.e. Let (x_s) and (y_s) be computable sequences of rational numbers which converge to x and y, respectively and witness the reduction $x \leq_S y$. That is, $|x - x_s| \leq c(|y - y_s| + 2^{-s})$ hold for all s and some constant c. By Lemma 3.2, we can assume that (y_s) is increasing. Thus we have

$$|x - x_s| \leq c(y - y_s + 2^{-s}) = c\left(\sum\nolimits_{i \geq s}(y_{s+1} - y_s) + 2^{-s}\right)$$
$$= \sum\nolimits_{i \geq s} c\left(y_{s+1} - y_s + 2^{-(i+1)}\right)$$

for all $s \in \mathbb{N}$. That is, the computable sequence (δ_s) of rational numbers defined by $\delta_s := \sum_{i \geq s} c(y_{s+1} - y_s + 2^{-(i+1)})$ satisfies condition (4). By Theorem 2.3, x is a d-c.e. real. Therefore, we have $S(\leq y) \subseteq \mathbf{WC}$.

Analogously to Theorem 1.1 we have

Corollary 3.8. *A d-c.e. real is random iff it is Solovay complete for* **WC** *iff it is an Ω-number.*

4 Solovay Reducibility on the Class DBC

We have shown that the class of d-c.e. reals has a Solovay-complete element. It is natural to ask, how about the larger classes? In this section, we consider the class **DBC** (divergence bounded computable) reals which is introduced by the authors in [13] and extends the class of d-c.e. reals properly. We show that **DBC** does not have a Solovay-complete element. Actually, we prove that **DBC** does not have a complete element even for the more weaker reducibility $\leq_{rH}$ of Downey, Hirschfeldt and LaForte [6].

Let's begin with the precise definitions of the notions mentioned above.

Definition 4.1 (Zheng, Rettinger and Gengler [13]). *A real x is called* divergence bounded computable *(dbc for short) if there is a computable function $h : \mathbb{N} \to \mathbb{N}$ and a computable sequence (x_s) of rational numbers which converges to x h-*bounded effectively *(h-b.e.) in the sense that there are at most $h(n)$ non-overlapping index-pairs (i, j) such that $|x_i - x_j| \geq 2^{-n}$ for any n.*

It is shown in [13] that the class **DBC** of all dbc reals is the closure of **WC** under the total computable real functions and it extends the class **WC** properly.

Definition 4.2 (Downey, Hirschfeldt and LaForte [6]). *A real x is* relative H reducible *to y ($x \leq_{rH} y$) if there is a constant j and a partial computable function $f :\subseteq \{0,1\}^* \times \mathbb{N} \to \{0,1\}^*$ such that*

$$(\forall n \in \mathbb{N})(\exists k \leq j)(f(y \upharpoonright n, k) = x \upharpoonright n). \tag{9}$$

The authors have shown in [14] that $x \leq_S y$ implies $x \leq_{rH} y$ for any c.a. reals x and y and there does not exist rH-complete element for the class of c.a. reals. The next theorem shows that **DBC** does not have rH-complete elements.

Theorem 4.3. *For any dbc real y there exists a dbc real x such that $x \not\leq_{rH} y$.*

Proof. For any divergence bounded computable real y, let h be a computable function and let (y_s) be a computable sequence of rational numbers which converges to y h-bounded effectively. We construct a computable sequence (x_s) of rational numbers which converges to x f-bounded effectively for a computable function f such that $x \not\leq_{rH} y$. That is, x satisfies, for all $i, j \in \mathbb{N}$, the following requirements:

$$R_{\langle i,j\rangle}: \qquad (\exists n)(\forall k \leq j)(\varphi_i(y \upharpoonright n, k) \neq x \upharpoonright n),$$

where (φ_i) is a computable enumeration of all partial computable functions $\varphi_i :\subseteq \{0,1\}^* \times \mathbb{N} \to \{0,1\}^*$.

Let (n_e) be an increasing computable sequence of natural numbers defined inductively by $n_{-1} := 0$ and $n_e := n_{e-1} + 2\pi_2(e) + 2$, where π_2 is the computable second inverse function of the paring function $\langle \cdot, \cdot \rangle$, i.e., $\pi_2(\langle i, j\rangle) = j$ for all $i, j \in \mathbb{N}$. To satisfy the requirement R_e for $e := \langle i, j\rangle$, we preserve the interval $[n_{e-1}, n_e)$ exclusively for R_e and define the rational number x_s in such a way that $x_s[n_{e-1} + 2k] \neq \varphi_{i,s}(y_s \upharpoonright n_e, k)[n_{e-1} + 2k]$, whenever $\varphi_{i,s}(y_s \upharpoonright n_e, k)$ is defined for some $k \leq j$, where $x[k]$ denotes the k-th symbol of the binary word x and $\varphi_{i,s}$ denotes the computable approximation of φ_i up to stage s (e.g., the portion of φ_i computed by a Turing machine in s stages). Since (y_s) converges, $y_s \upharpoonright n_e$ changes only finitely often and the constructed sequence (x_s) converges too. Obviously, the limit $x := \lim_{s\to\infty} x_s$ satisfies all requirements R_e.

Unfortunately, the real x constructed in this way is not necessarily divergence bounded computable, because the number of changes of $y_s \upharpoonright n_e$ is not computably bounded in general and each change of $y_s \upharpoonright n_e$ forces a change of $x_s \upharpoonright n_e$. However, since (y_s) converges h-bounded effectively, the number of its jumps larger than 2^{-n_e} is bounded above by $h(n_e)$. In other words, what we do not computably bound is the number of relatively small jumps, i.e., the jumps (s, t) such that $|y_s - y_t| < 2^{-n_e}$. But for such kind of small jumps, we have either $y_s \upharpoonright n_e = w011\cdots1$ and $y_t \upharpoonright n_e = w100\cdots0$ or $y_s \upharpoonright n_e = w100\cdots0$ and $y_t \upharpoonright n_e = w011\cdots1$. If such small jumps occur successively, $y_s \upharpoonright n_e$ takes the values $w011\cdots1$ and $w100\cdots0$ in turn. Thus, it suffices to define x_s in such a way that $x_s \upharpoonright n_e$ differs from both $\varphi_i(w100\cdots0, k)$ and $\varphi_i(w011\cdots1, k)$ (for $k \leq j$) to avoid unnecessary changes of x_s. This leads to the following revised construction.

At any stage s, if y_s makes a jump of a size at least 2^{-n_e}, then we define x_s in the way mentioned above, i.e., $x_s[n_{e-1} + 2k] \neq \varphi_{i,s}(y_s \upharpoonright n_e, k)[n_{e-1} + 2k]$ whenever $\varphi_{i,s}(y_s \upharpoonright n_e, k)$ is defined for some $k \leq j$. Otherwise, suppose that $|y_s - y_t| < 2^{-n_e}$ for the last stage t before s at which $y_t \upharpoonright n_e$ changes. Then we define x_s in such a way that the two bits $x_s[n_{e-1} + 2k, n_{e-1} + 2k + 1]$ does not equal to $\varphi_{i,s}(y_v \upharpoonright n_e, k)[n_{e-1} + 2k, n_{e-1} + 2k + 1]$ for $v = s, t$, where

$x[i,j] := x[i]x[i+1]\cdots x[j]$ for any $i \leq j$. Thus, if y_s makes jumps less than 2^{-n_e} successively, x_s has to be changed only once. Now, for the maximal index-chain $t_0 < s_0 \leq t_1 < s_1 \leq \cdots \leq t_m < s_m$ such that $|y_{t_v} - y_{s_v}| \geq 2^{-n_e}$ for all $v \leq m$, the segment $x_s[n_{e-1}, n_e]$ may change for $s = s_0, s_1, \ldots, s_m$. Between t_v and s_v or between s_v and t_{v+1} or after s_m, $x_s[n_{e-1}, n_e]$ can change at most once for each. That is, $x_s[n_{e-1}, n_e]$ can change at most $3h(n_e)$ times totaly because $m \leq h(n_e)$. This implies that the sequence (x_s) converges f-bounded effectively for a computable function f and hence the limit x is dbc.

Corollary 4.4. *The class* **DBC** *does not have Solovay-complete elements.*

References

1. K. Ambos-Spies, K. Weihrauch, and X. Zheng. Weakly computable real numbers. *Journal of Complexity*, 16(4):676–690, 2000.
2. C. S. Calude, P. H. Hertling, B. Khoussainov, and Y. Wang. Recursively enumerable reals and Chaitin Ω numbers. *Theoretical Computer Science*, 255:125–149, 2001.
3. G. Chaitin. A theory of program size formally identical to information theory. *J. of ACM.*, 22:329–340, 1975.
4. R. G. Downey. Some recent progress in algorithmic randomness. In *MFCS'04*, pages 42–83, 2004.
5. R. G. Downey and D. R. Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer-Verlag, 200? monograph to be published.
6. R. G. Downey, D. R. Hirschfeldt, and G. LaForte. Randomness and reducibility. In J. Sgall, A. Pultr, and P. Kolman, editors, *MFCS 2001, Mariánské Lázně, Czech Republic, August 27-31, 2001*, volume 2136 of *LNCS*, pages 316–327. Springer, 2001.
7. R. G. Downey, D. R. Hirschfeldt, and A. Nies. Randomness, computability, and density. *SIAM J. Comput.*, 31(4):1169–1183 (electronic), 2002.
8. A. Kuçera and T. A. Slaman. Randomness and recursive enumerability. *SIAM J. Comput.*, 31(1):199–211, 2001.
9. L. A. Levin. The concept of a random sequence. *Dokl. Akad. Nauk SSSR*, 212:548–550, 1973. (English translation: Soviet Math. Dokl. 212 (1973), 1413–1416 (1974)).
10. P. Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.
11. A. Raichev. D.c.e. reals, relative randomness, and real closed fields. In *CCA 2004, August 16-20, 2004, Lutherstadt Wittenberg, Germany*, 2004.
12. R. M. Solovay. Draft of a paper (or a series of papers) on chaitin's work .... manuscript, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, p. 215, 1975.
13. X. Zheng, R. Rettinger, and R. Gengler. Closure properties of real number classes under CBV functions. *Theory of Computing Systems*, 2005. (to appear).
14. X. Zheng and R. Rettingre. On the extensions of solovay reducibility. In *COOCON 2004, August 17-20, Jeju Island, Korea*, volume 3106 of *LNCS*. Springer-Verlage, 2004.

Algorithms for Terminal Steiner Trees*

Fábio Viduani Martinez[1], José Coelho de Pina[2], and José Soares[2]

[1] Universidade Federal de Mato Grosso do Sul, Brazil
fhvm@dct.ufms.br
[2] Universidade de São Paulo, Brazil
{coelho,jose}@ime.usp.br

Abstract. The terminal Steiner tree problem (TST) consists of finding a minimum cost Steiner tree where each terminal is a leaf. We describe a factor $2\rho - \rho/(3\rho - 2)$ approximation algorithm for the TST, where ρ is the approximation factor of a given algorithm for the Steiner tree problem. Considering the current best value of ρ, this improves a previous 3.10 factor to 2.52. For the TST restricted to instances where all edge costs are either 1 or 2, we improve the approximation factor from 1.60 to 1.42.

1 Introduction

Consider a graph G and a function c from its edge set into $\mathbb{Q}_{\geq}$. By V_G and E_G we denote the vertex and edge sets of G, respectively. For any subset F of E_G let $c(F) := \sum_{e \in F} c(e)$ and for any subgraph H of G let $c(H) := c(E_H)$ be its cost.

Given a graph and a subset of vertices called *terminals*, a *Steiner tree* is a connected subgraph that contains all terminals. A vertex which is not a terminal is called a *Steiner vertex*. The *Steiner tree problem* (ST) is the following.

> **Problem** ST(G, c, R): *given a complete graph G, a cost function $c\colon E_G \to \mathbb{Q}_{\geq}$ satisfying the triangle inequality, and a set $R \subseteq V_G$ of terminals, find a minimum cost Steiner tree.*

We shall denote by ρ the approximation factor of a given approximation algorithm for the ST. Currently the best value for ρ is slightly smaller than 1.55 [8].

A Steiner tree is a *terminal Steiner tree* if its set of leaves is precisely the set of terminal vertices. Terminal Steiner trees have an important role in applications such as construction of phylogenetic trees in biology [7], global and local routing in VLSI-design [3, 4, 6], transportation and telecommunications [4, 6]. The *terminal Steiner tree problem* (TST) is as follows.

> **Problem** TST(G, c, R): *given a complete graph G, a cost function $c\colon E_G \to \mathbb{Q}_{\geq}$ satisfying the triangle inequality, and a set $R \subseteq V_G$ of terminals, find a minimum cost terminal Steiner tree.*

The first result for the TST was obtained by Lin and Xue [6]. They proposed a factor $2 + \rho$ approximation algorithm. Fuchs [4], Chen, Lu and Tang [2], and Drake and

* This work was supported by FAPESP/CNPq (ProNEx project 2003/09925-5).

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 369–379, 2005.

Hougardy [3], obtained independently a factor 2ρ approximation algorithm for the TST. We present an approximation algorithm for the TST, which improves the 2ρ factor.

Theorem 1 *There is a factor α approximation algorithm for the* TST*, where*

$$\alpha = 2\rho - \rho/(3\rho - 2)$$

and ρ is the approximation factor of a given algorithm for the ST.

Since the best currently value for ρ is about 1.55, Theorem 1 implies the following corollary.

Corollary 2 *There is a factor* 2.52 *approximation algorithm for the* TST.

Let $\mathrm{ST}_{1,2}$ denote the ST restricted to instances where all edge costs are either 1 or 2. Bern and Plassmann [1] showed the **Max-SNP**-hardness of the $\mathrm{ST}_{1,2}$ and proposed a factor 4/3 approximation algorithm. Robins and Zelikovsky [8] improved this factor to 1.28. Let $\mathrm{TST}_{1,2}$ denote the TST restricted to instances where all edge costs are either 1 or 2. Lu, Tang and Lee [7] proposed an 8/5 factor approximation algorithm for the $\mathrm{TST}_{1,2}$ and showed that the problem is **Max-SNP**-hard. We are able to show the following result.

Theorem 3 *There is a factor* 17/12 *approximation algorithm for the* $\mathrm{TST}_{1,2}$.

2 Factor α Algorithm for the TST

Let G, c, and R be an instance for the TST. For each r in R we denote by e_r a minimum cost edge connecting r to $V_G \setminus R$ and we denote by F_R the set $\{e_r : r \in R\}$. We describe an approximation algorithm that receives G, c, and R and constructs terminal Steiner trees T_1 and T_2 such if $c(F_R)$ is 'small' compared to $\mathrm{opt}(\mathrm{TST}(G, c, R))$ then

$$c(T_1) \leq \alpha\, \mathrm{opt}(\mathrm{TST}(G, c, R))\,, \quad \text{otherwise} \quad c(T_2) \leq \alpha\, \mathrm{opt}(\mathrm{TST}(G, c, R))\,.$$

The algorithm returns the cheapest tree constructed.

Throughout this section we make a few assumptions without loss of generality. As the TST is trivial for $|R| \leq 2$, we assume $|R| \geq 3$. For $|R| \geq 3$ a TST contains no edge connecting terminals, so we may assume that for each pair r, s of vertices in R the cost on the edge rs is $\min_{v \in V_G \setminus R}(c(rv) + c(vs))$. In particular, this last assumption implies that we may also assume that any Steiner tree for $\mathrm{ST}(G, c, R)$ contains no edge connecting terminals.

The lemma below presents a situation where $\mathrm{opt}(\mathrm{TST}(G, c, R)) = \mathrm{opt}(\mathrm{ST}(G, c, R))$.

Lemma 1. *Let G, c, and R be an instance for the* TST*. Suppose that for each edge $e_r = ru$, r in R, and for each vertex w in $V_G \setminus \{r\}$ we have that $c(uw) \leq c(rw) - c(ru)/2$. Then, given a Steiner tree S we can find in polynomial time a terminal Steiner tree T such that $c(T) \leq c(S)$.*

Proof. If each vertex in R is a leaf in S, then we take $T := S$ and we are done. So, we may assume that there exists a vertex r in R such that its set W of neighbors in S has more than 1 vertex. Let u be the vertex such that $e_r = ru$, let $X := \{rw : w \in W\}$, and let $Y := \{uw : w \in W\}$ (Figure 1). Finally, let S' be the Steiner tree induced by the edges in $(E_S \setminus X) \cup Y \cup \{ru\}$. We have that

$$\begin{aligned} c(S') &= c(E_S) - c(X) + c(Y) + c(ru) \\ &\leq c(E_S) - c(X) + c(X) - |W|c(ru)/2 + c(ru) \qquad (1) \\ &= c(E_S) + (1 - |W|/2)c(ru) \\ &\leq c(E_S) = c(S), \end{aligned}$$

where (1) holds because $c(uw) \leq c(rw) - c(ru)/2$ for each w in W. In this process we obtained a Steiner tree where r is a leaf and all terminal that are leaves in S remain leaves in S'. Thus, by repeating this process we eventually end up with a terminal Steiner tree satisfying the lemma. □

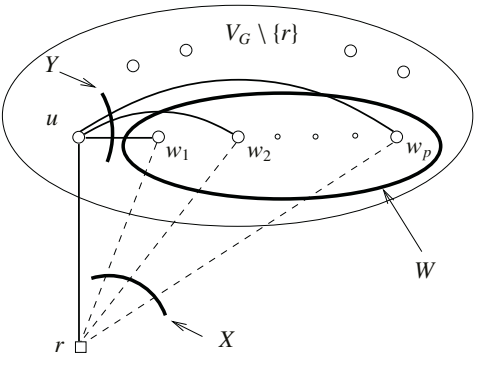


Fig. 1. Illustration for Lemma 1. Dashed edges are removed and solid edges are included. Vertex u may belong to W.

Lemma 2. *Let G, c, and R be an instance for the* TST *and let $F_R := \{e_r : r \in R\}$. Given a Steiner tree S we can find in polynomial time a terminal Steiner tree T such that $c(T) = 2c(S) - c(F_R)$.*

Proof. If each vertex in R is a leaf in S, then we take $T := S$ and we are done. So, we may assume that there exists a vertex r in R such that its set W of neighbors in S has more than 1 vertex. Let u be the vertex such that $e_r = ru$, let $X := \{rw : w \in W\}$, let rx and ry be the two most costly edges in X and let Y be the edge set of a xy-path in the subgraph induced by the vertices of W (Figure 2). By the triangle inequality,

$$c(Y) \leq 2c(X) - c(rx) - c(ry). \qquad (2)$$

Finally, let S' be the Steiner tree induced by the edges in $(E_S \setminus X) \cup Y \cup \{rz\}$, where rz is the cheapest edge in X. We have that

$$\begin{aligned} c(S') &= c(E_S) - c(X) + c(Y) + c(rz) \\ &\leq c(E_S) - c(X) + 2c(X) - c(rx) - c(ry) + c(rz) \qquad (3) \\ &\leq c(E_S) + c(X) - c(rx) \\ &\leq c(E_S) + c(X) - c(ru) = c(S) + c(X) - c(ru) \end{aligned}$$

where (3) follows from (2). S' is a tree where r is a leaf and all terminal that are leaves in S remain leaves. By repeating this process we eventually obtain a terminal Steiner tree satisfying the lemma. □

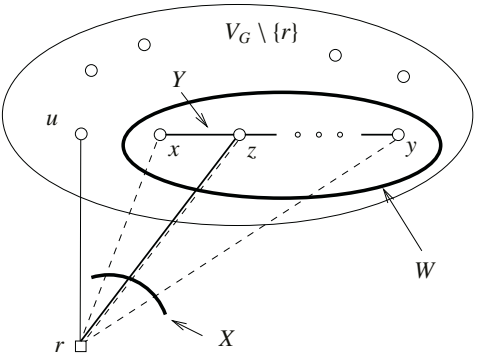


Fig. 2. Illustration for Lemma 2. Dashed edges are removed and solid edges are included. Vertex u may belong to W.

ALGORITHM TERMINAL: receives a complete graph G, a cost function $c\colon E_G \to \mathbb{Q}_{\geq}$ satisfying the triangle inequality, and a set $R \subseteq V_G$ of terminals, and returns a terminal Steiner tree T such that $c(T) \leq \alpha\,\mathrm{opt}(\mathrm{TST}(G, c, R))$.

1: Let $c' : E_G \to \mathbb{Q}_{\geq}$ defined by

$$c'(e) := \begin{cases} c(e) + 3c(e_r) + 3c(e_s)\,, & \text{if } e \cap R = \{r, s\} \\ c(e) + 3c(e_r)\,, & \text{if } e \cap R = \{r\} \\ c(e)\,, & \text{if } e \cap R = \emptyset. \end{cases}$$

2: Let S_1 be a Steiner tree returned by a factor ρ approximation algorithm for $\mathrm{ST}(G, c', R)$.
3: Construct a terminal Steiner tree T_1 from S_1 using Lemma 1.
4: Let S_2 be the Steiner tree returned by a factor ρ approximation algorithm for $\mathrm{ST}(G, c, R)$.
5: Construct a terminal Steiner tree T_2 from S_2 using Lemma 2.
6: Let T the minimum cost tree between T_1 and T_2.
7: Return T and stop.

Proof of Theorem 1. Let c', S_1, S_2, T_1 and T_2 be the cost function, Steiner trees and terminal Steiner trees produced by the ALGORITHM TERMINAL to an instance G, c and R. Suppose firstly that $c(F_R) \leq (\rho/(3\rho - 2))\ \mathrm{opt}(\mathrm{TST}(G, c, R))$. As c' fulfills the hypothesis of Lemma 1, then $c'(T_1) \leq c'(S_1)$. Moreover, for each r in R the edge incident to r in T_1 is e_r. Hence,

$$\begin{aligned} c(T_1) &= c'(T_1) - 3c(F_R) \\ &\leq c'(S_1) - 3c(F_R) \\ &\leq \rho \operatorname{opt}(\mathrm{ST}(G, c', R)) - 3c(F_R) \\ &\leq \rho \left(\operatorname{opt}(\mathrm{TST}(G, c, R)) + 3c(F_R)\right) - 3c(F_R) \qquad (4) \\ &= \rho \operatorname{opt}(\mathrm{TST}(G, c, R)) + 3(\rho - 1)c(F_R) \\ &\leq \alpha \operatorname{opt}(\mathrm{TST}(G, c, R)) , \end{aligned}$$

where (4) holds because if T^* is a terminal Steiner such that $c(T^*) = \operatorname{opt}(\mathrm{TST}(G, c, R))$, then

$$\operatorname{opt}(\mathrm{ST}(G, c', R)) \leq c'(T^*) = c(T^*) + 3c(F_R) = \operatorname{opt}(\mathrm{TST}(G, c, R)) + 3c(F_R) .$$

Now, suppose that $c(F_R) \geq (\rho/(3\rho - 2))\ \operatorname{opt}(\mathrm{TST}(G, c, R))$. Let S_2 be the Steiner tree constructed by the algorithm. We have that

$$\begin{aligned} c(T_2) &\leq 2c(S_2) - c(F_R) \qquad (5) \\ &\leq 2\rho\ \operatorname{opt}(\mathrm{ST}(G, c, R)) - c(F_R) \\ &\leq 2\rho \operatorname{opt}(\mathrm{TST}(G, c, R) - c(F_R) \\ &\leq \alpha \operatorname{opt}(\mathrm{TST}(G, c, R)) , \end{aligned}$$

where (5) holds because of Lemma 2.

Finally, as the ALGORITHM TERMINAL returns the cheapest tree T between T_1 and T_2, we conclude that $c(T) \leq \alpha\ \operatorname{opt}(\mathrm{TST}(G, c, R))$. □

3 Factor 17/12 Algorithm for $\mathrm{TST}_{1,2}$

We describe an approximation algorithm that receives G, c and R, an instance for $\mathrm{TST}_{1,2}$ such that $c\colon E_G \to \{1, 2\}$, and constructs a terminal Steiner tree T such that $c(T) \leq (17/12) \operatorname{opt}(\mathrm{TST}_{1,2}(G, c, R))$.

Throughout this section we assume that the given instance is feasible and that $|R| \geq 3$. The algorithm constructs a tree T_v for each $v \in V_G \setminus R$. Each T_v is such that $c(T_v)$ is at most $17/12$ times the cost of any terminal Steiner tree using v as a Steiner vertex. The minimum cost tree over all T_v is returned by the algorithm. Since a minimum cost terminal Steiner tree must use at least one Steiner vertex, the tree returned is a $17/12$ approximation for $\mathrm{TST}_{1,2}$.

Proof of Theorem 3. We claim that ALGORITHM $\mathrm{TERMINAL}_{1,2}$ constructs a terminal Steiner with the desired approximation.

The algorithm constructs, for each $v \in V_G \setminus R$, a terminal Steiner tree T_v containing v as a Steiner vertex. As a straightforward consequence of Lemmas 6, 4, and 3 presented below, $c(T_v)$ is at most $17/12$ times the cost of a minimum cost terminal Steiner tree containing v.

If the instance (G, c, R) is feasible and $|R| \geq 3$, any terminal Steiner tree contains at least one Steiner vertex v. Therefore, T_v is the desired approximation for the given instance of $\mathrm{TST}_{1,2}$. □

ALGORITHM TERMINAL$_{1,2}$: receives a complete graph G, a cost function $c\colon E_G \to \{1,2\}$, and a set $R \subseteq V_G$ of terminals, and returns a terminal Steiner tree.

1: For each $v \in V_G \setminus R$
2: Construct a set R_1 using REDUCTION 1 on (G, c, R, v).
3: Construct sets R_2 and S using REDUCTION 2 on $(G, c, R \setminus R_1)$.
4: Construct a terminal tree T''_v using ALGORITHM CENTRAL on $(G, c, R \setminus (R_1 \cup R_2), v)$.
5: Construct a terminal tree T'_v for $(G, c, R \setminus R_1)$ using tree T''_v, set S and Lemma 3.
6: Construct a terminal tree T_v for (G, c, R, v) using tree T'_v and Lemma 4.
7: Let T be a minimum cost tree among the trees constructed in Step 1.
8: Return T and stop.

Reductions

The core of ALGORITHM TERMINAL$_{1,2}$ is ALGORITHM CENTRAL. But, before using ALGORITHM CENTRAL, we use two reductions to remove some vertices from R. Here we describe these reductions. The lemmas below show how to obtain, afterwards, the terminal Steiner tree for the whole R keeping the approximation achieved by ALGORITHM CENTRAL.

The idea of REDUCTION 1 is that if we assume that a Steiner vertex v is part of the terminal Steiner tree, then every cost 1 edge connecting v to a terminal vertex can be part of a minimum cost terminal Steiner tree. The fraction 17/12 appearing in the statement of Lemma 3 can be replaced by any value greater or equal to 1.

REDUCTION 1: receives a complete graph G, a cost function $c\colon E_G \to \{1,2\}$, a set $R \subseteq V_G$ of terminals, and a vertex $v \in V_G \setminus R$, and returns a set $R' \subseteq R$.

1: Let R' be the set $R' := \{r : r \in R \text{ and } c(rv) = 1\}$.
2: Return R' and stop.

Let G, c and R be an instance of TST$_{1,2}$, and let v be a vertex in $V_G \setminus R$. We denote by $\mathrm{opt}(G, c, R, v)$ the cost of a minimum cost terminal Steiner tree containing v.

Lemma 3. *Given the set R' constructed by* REDUCTION 1 *and a terminal Steiner tree T_1 for $(G, c, R \setminus R')$ containing v such that*

$$c(T_1) \leq (17/12)\,\mathrm{opt}(G, c, R \setminus R', v) \tag{6}$$

we can find in polynomial time a terminal Steiner tree T_2 for (G, c, R) such that

$$c(T_2) \leq (17/12)\,\mathrm{opt}(G, c, R, v).$$

Proof. Omitted (see appendix A). □

The idea behind REDUCTION 2 is that a Steiner vertex connected to several terminal vertices by cost 1 edges is a good vertex to be part of a terminal Steiner tree. Here, several means at least 5. The fraction 17/12 appearing in the statement of Lemma 4 can be replaced by any value greater or equal to 7/5. A set S is constructed to remember which Steiner vertices were considered during the reduction.

The work done by ALGORITHM CENTRAL would be made easier if we replace 5 by 4 in the reduction. However, it is easy to check that doing so the reduction will achieve approximation factor of $3/2$ instead of $7/5$. Since $3/2 > 17/12$, the main result of this section would not hold.

REDUCTION 2: receives a complete graph G, a cost function $c: E_G \to \{1, 2\}$, a set $R \subseteq V_G$ of terminals, and returns sets $R' \subseteq R$ and $S \in V_G \setminus R$.

1: $R' := \emptyset$
2: $S := \emptyset$
3: For each $s \in V_G \setminus R$ let $N(s) := \{r : r \in R \text{ and } c(rs) = 1\}$.
4: If there exists $s \in V_G \setminus R$ such that $|N(s)| \geq 5$, then let $R' := R' \cup N(s)$, $S = S \cup \{s\}$, $R := R \setminus N(s)$, and return to Step 3.
5: Return (R', S) and stop.

Lemma 4. *Given the sets R' and S constructed by* REDUCTION 2 *and a terminal Steiner tree T_1 for $(G, c, R \setminus R')$ containing a Steiner vertex v such that*

$$c(T_1) \leq (17/12)\,\mathrm{opt}(G, c, R \setminus R', v) \tag{7}$$

we can find in polynomial time a terminal Steiner tree T_2 for (G, c, R) such that

$$c(T_2) \leq (17/12)\,\mathrm{opt}(G, c, R, v).$$

Proof. Omitted (see appendix A). □

Algorithm Central

Similarly to REDUCTION 2, this algorithm tries to use Steiner vertices connected by cost 1 edges to several terminal vertices. Here, several means 4 or 3. We compare the number of such Steiner vertices that we can find to the number that the minimum cost terminal Steiner tree can use. This leads us to a weighted set packing problem, as described below. Let $\mathcal{C}$ be a collection of sets. We remember that a *set packing* is a collection of mutually disjoint sets in $\mathcal{C}$.

ALGORITHM CENTRAL: receives a complete graph G, a cost function $c: E_G \to \{1, 2\}$, a set $R \subseteq V_G$ of terminals, and a vertex $v \in V_G \setminus R$. The algorithm returns a terminal Steiner tree.

1: For each $s \in V_G \setminus R$ let $N(s)$ be the set $N(s) := \{r : r \in R \text{ and } c(rs) = 1\}$.
2: Let $\mathcal{C}$ be the collection of sets $\mathcal{C} := \{N(s) : s \in V_G \setminus R\} \cup \{C : C \subseteq N(s) \text{ for some } s, \text{ and } |C| = 3\}$
3: Construct a set packing $\mathcal{A}$ of $\mathcal{C}$ using Lemma 5.
4: For each $A \in \mathcal{A}$, choose a unique $v(A) \in V_G \setminus R$ such that $A \subset N(v(A))$.
5: Let V_T be the set of vertices $V_T := \{v\} \cup \{R\} \cup \{v(A) : A \in \mathcal{A}\}$
6: Let E_T be the set of edges $E_T := \{rs : r \text{ is covered by } \mathcal{A} \text{ and } s = v(A) \text{ for some } A \in \mathcal{A}\} \cup \{sv : s = v(A) \text{ for some } A \in \mathcal{A}\} \cup \{rv : r \text{ is not covered by } \mathcal{A}\}$.
7: Return $T = (V_T, E_T)$ and stop.

Lemma 5. *Let $\mathcal{C}$ be a collection of sets such that each set $C \in \mathcal{C}$ has 3 or 4 elements and such that if C has 4 elements, then each 3-subset of C is also in $\mathcal{C}$. We can find*

in polynomial time a set packing $\mathcal{A}$ of C with a_4 sets of cardinality 4 *and a_3 sets of cardinality* 3*, such that*

$$(8b_4 + 6b_3 - 2a_4 - a_3)/(5b_4 + 4b_3) \leq 17/12.$$

where b_4 and b_3 are the sets of cardinality 4 *and* 3*, respectively, in an arbitrary non-empty set packing $\mathcal{B}$ of C.*

Proof. We begin packing sets of size 4. Here we use an algorithm due to Hurkens and Schrijver [5]. Their algorithm guarantees that, for any fixed ϵ, we can find in polynomial time a packing with at least $b_4/(2+\epsilon)$ sets. We just need a packing with a_4 sets of size 4, where

$$a_4 \geq 3b_4/8. \tag{8}$$

Besides these a_4 sets, we greedily add to $\mathcal{A}$ as many sets with 3 elements as possible. Let $\mathcal{B}_4$ and $\mathcal{B}_3$ be the collection of sets of size 4 and 3, respectively, in $\mathcal{B}$.

As a consequence of the greedy choice of sets of size 3 in $\mathcal{A}$, we notice that each set in $B \in \mathcal{B}_4$ has at least 2 elements covered by $\mathcal{A}$, since, otherwise, a 3-subset of B would be added to $\mathcal{A}$. Also, each set in $B \in \mathcal{B}_3$ has at least 1 element covered by $\mathcal{A}$, since, otherwise, B would be added to $\mathcal{A}$. So, since each element is covered at most once, the total number of elements covered by $\mathcal{A}$ is at least $2b_4 + b_3$.

On the other hand, $\mathcal{A}$ can cover at most $4a_4 + 3a_3$ elements. Therefore, it holds that

$$4a_4 + 3a_3 \geq 2b_4 + b_3. \tag{9}$$

Using inequality (9) and the hypothesis that $\mathcal{B}$ is non-empty, we obtain that

$$\begin{aligned}(8b_4 + 6b_3 - 2a_4 - a_3)/(5b_4 + 4b_3) &= (24b_4 + 18b_3 - 6a_4 - 3a_3)/(15b_4 + 12b_3)\\ &\leq (24b_4 + 18b_3 - 2a_4 - 2b_4 - b_3)/(15b_4 + 12b_3)\\ &= (22b_4 + 17b_3 - 2a_4)/(15b_4 + 12b_3).\end{aligned}$$

Now, using inequality (8) we obtain that

$$\begin{aligned}(22b_4 + 17b_3 - 2a_4)/(15b_4 + 12b_3) &\leq (22b_4 + 17b_3 - 3b_4/4)/(15b_4 + 12b_3)\\ &= (85b_4/4 + 17b_3)/(15b_4 + 12b_3) = 17/12.\end{aligned}$$

□

Lemma 6. *For every instance (G, c, R, v) such that no further reduction in R is possible using Reductions 1 and 2,* Algorithm Central *produces a terminal Steiner tree T such that*

$$c(T) \leq (17/12)\,\mathrm{opt}(G, c, R, v).$$

Proof. Let T^* be a minimum cost terminal Steiner tree containing vertex v. Let B_4 be the set of Steiner vertices connected in T^* to exactly 4 terminal vertices using cost 1 edges. Let B_3 and B_2 be defined in a similar way. Let b_4, b_3, and b_2 be the cardinality

of B_4, B_3, and B_2, respectively. In what follows, we consider T^* as a rooted tree with v as a root.

Let s be a vertex in B_4. We can associate cost 5 for s: there are at least 4 edges connecting s to a terminal vertex and 1 edge connecting s to its ancestral in T^*. Similarly, we can associate cost 4 and 3 for vertices in B_3 and B_2, respectively. Notice that the edges mentioned above are disjoint.

We have considered the cost of connecting exactly $4b_4 + 3b_3 + 2b_2$ terminal vertices in T^*. Consider a remaining terminal vertex r. We argue now that we can account cost 2 for each one of such vertex. Either *(i)* r is connected in T^* by a cost 2 edge or *(ii)* r is connected to a Steiner vertex s by a cost 1 edge. If *(i)* happens, then we account cost 2 due to the edge connecting r to the tree. Suppose that *(ii)* happens. Notice that, since REDUCTION 1 was performed, $s \neq v$. Therefore, s has an ancestral in T^*. Also, since REDUCTION 2 was performed, s connects at most 4 terminal vertices by cost 1 edges. But, since r is not one of the $4b_4 + 3b_3 + 2b_2$ terminal vertices, we infer that r is the unique terminal vertex connected to s by cost 1 edges. So, we can account 1 for the edge rs and 1 for the edge connecting s to its ancestral in T^*.

So, we have that

$$c(T^*) \geq 5b_4 + 4b_3 + 3b_2 + 2b_1, \tag{10}$$

where $b_1 = |R| - (4b_4 + 3b_3 + 2b_2)$.

Now, we consider the cost of the tree constructed by ALGORITHM CENTRAL. The construction is based on a set packing $\mathcal{A}$, where for each $A \in \mathcal{A}$ corresponds a Steiner vertex $v(A)$.

The set E_T defined in the algorithm can be partitioned into E_1, E_2, and E_3, where $E_1 := \{rs : r \text{ is covered by } \mathcal{A} \text{ and } s = v(A) \text{ for some } A \in \mathcal{A}\}$, $E_2 := \{sv : s = v(A) \text{ for some } A \in \mathcal{A}\}$, and $E_3 := \{rv : r \text{ is not covered by } \mathcal{A}\}$. Let a_4 and a_3 be the number of sets in $\mathcal{A}$ with 4 and 3 elements, respectively. The cost of E_1 is $c(E_1) = |E_1| = 4a_4 + 3a_3$. The cost of E_2 is $c(E_2) \leq 2|E_2| = 2(a_4 + a_3)$. The cost of E_3 is $c(E_3) \leq 2|E_3| = 2(|R| - (4a_4 + 3a_3))$. Summarizing, we have that

$$c(T) \leq 6a_4 + 5a_3 + 2a_2, \tag{11}$$

where $a_2 = |R| - (4a_4 + 3a_3)$.

Notice also that the number of terminals in R can be expressed by

$$4b_4 + 3b_3 + 2b_2 + b_1 = 4a_4 + 3a_3 + a_2. \tag{12}$$

In the following, we assume that all the denominators are non zero, since, otherwise, the analysis becomes easier. Hence, we have that

$$\frac{c(T)}{c(T^*)} \leq \frac{6a_4 + 5a_3 + 2a_2}{5b_4 + 4b_3 + 3b_2 + 2b_1} \tag{13}$$

$$= \frac{6a_4 + 5a_3 + 2(4b_4 + 3b_3 + 2b_2 + b_1 - 4a_4 - 3a_3)}{5b_4 + 4b_3 + 3b_2 + 2b_1} \tag{14}$$

$$= \frac{8b_4 + 6b_3 + 4b_2 + 2b_1 - 2a_4 - a_3}{5b_4 + 4b_3 + 3b_2 + 2b_1}$$

$$\leq \max\left\{\frac{8b_4 + 6b_3 - 2a_4 - a_3}{5b_4 + 4b_3}, \frac{4b_2 + 2b_1}{3b_2 + 2b_1}\right\} \leq \frac{17}{12} \tag{15}$$

where inequality (13) follows from inequalities (11) and (10), equality (14) follows from equality (12), and the last inequality follows from Lemma 5. □

Remark

One can slightly improve the $17/12 = 1.416\ldots$ bound of Lemma 5 to a value asymptotically close to $38/27 = 1.407\ldots$ To achieve this bound,

1. apply Hurkens and Schrijver [5] algorithm to obtain a packing $\mathcal{A}'$ with
$$a'_4 \geq b_4/(2+\epsilon)$$
sets of size 4, which is greedily extended to a packing of C by adding sets of size 3;
2. apply Hurkens and Schrijver algorithm to C restricted to sets of size 3 to obtain a packing $\mathcal{A}''$ with
$$a''_3 \geq 2(b_4 + b_3)/(3+\epsilon')$$
sets of size 3;
3. choose as result the packing which maximizes $2a_4 + a_3$, where a_4 and a_3 are the numbers of sets with 4 and 3 elements, respectively.

This, in turn, improves the factor of Theorem 3 to a value asymptotically close to $38/27$.

Acknowledgement

We would like to thank an anonymous referee for helpful comments in a preliminary version of this work.

References

1. M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32:171–176, 1989.
2. Y. H. Chen, C. L. Lu, and C. Y. Tang. On the full and bottleneck full Steiner tree problems. In *The Ninth International Computing and Combinatorics Conference – COCOON*, pages 122–129, Big Sky, MT, USA, July 2003.
3. D. E. Drake and S. Hougardy. On approximation algorithms for the terminal Steiner tree problem. *Information Processing Letters*, 89(1):15–18, 2004.
4. B. Fuchs. A note on the terminal Steiner tree problem. *Information Processing Letters*, 87:219–220, 2003.
5. C. A. J. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal of Discrete Mathematics*, 2(1):68–72, 1989.
6. G. Lin and G. Xue. On the terminal Steiner tree problem. *Information Processing Letters*, 84:103–107, 2002.
7. C. L. Lu, C. Y. Tang, and R. C.-T. Lee. The full Steiner tree problem. *Theoretical Computer Science*, 306:55–67, 2003.
8. G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 770–779, San Francisco, California, USA, January 2000.

A Omitted Proofs

Proof of Lemma 3. Notice that, since the removal of any terminal vertex from R decreases the cost of the minimum terminal Steiner tree by at least 1, we have that

$$\text{opt}(G, c, R \setminus R', v) \leq \text{opt}(G, c, R, v) - |R'|. \tag{16}$$

We construct T_2 from T_1 adding to T_1 some vertices and edges. We add to T_1 the vertices in R' to T_2 and the edges connecting vertices in R' to v. Since each one of these edges has cost 1, we have that

$$c(T_2) = c(T_1) + |R'|. \tag{17}$$

Using the hypothesis (6) and combining inequalities (16) and (17), we obtain that

$$\begin{aligned} c(T_2) &= c(T_1) + |R'| \\ &\leq (17/12)\,\text{opt}(G, c, R \setminus R', v) + |R'| \\ &\leq (17/12)\,(\text{opt}(G, c, R, v) - |R'|) + |R'| \\ &\leq (17/12)\,\text{opt}(G, c, R, v). \end{aligned}$$

□

Proof of Lemma 4. Notice that, since the removal of any terminal vertex from R decreases the cost of the minimum terminal Steiner tree by at least 1, we have that

$$\text{opt}(G, c, R \setminus R', v) \leq \text{opt}(G, c, R, v) - |R'|. \tag{18}$$

We construct T_2 from T_1 adding to T_1 some vertices and edges. We add to T_1 the vertices in $R' \cup S$. We add cost 1 edges connecting vertices in S to vertices in R'. If necessary, we add for each $s \in S$ the edge connecting s to v. It is easy to verify that T_2 is a terminal Steiner tree for (G, c, R). The cost of T_2 is

$$c(T_2) \leq c(T_1) + |R'| + 2|S|.$$

By construction, for each vertex in S there are at least 5 vertices in R'. Thus, we have that

$$5|S| \leq |R'|.$$

Combining the last two inequalities, we obtain that

$$c(T_2) \leq c(T_1) + 7|R'|/5. \tag{19}$$

Using the hypothesis (7) and inequalities (18) and (19), we obtain that

$$\begin{aligned} c(T_2) &= c(T_1) + 7|R'|/5 \\ &\leq (17/12)\,\text{opt}(G, c, R \setminus R', v) + 7|R'|/5 \\ &\leq (17/12)\,(\text{opt}(G, c, R, v) - |R'|) + 7|R'|/5 \\ &\leq (17/12)\,\text{opt}(G, c, R, v). \end{aligned}$$

□

Simple Distributed Algorithms for Approximating Minimum Steiner Trees

Parinya Chalermsook[1,2,⋆] and Jittat Fakcharoenphol[3,⋆⋆]

[1] Asian Institute of Technology, Pathumthanim, Thailand
[2] Kasetsart University, Bangkok, Thailand
fengpycs@ku.ac.th
[3] Department of Computer Engineering
Kasetsart University, Bangkok, Thailand
jtf@ku.ac.th

Abstract. Given a network $G = (V, E)$, edge weights $w(\cdot)$, and a set of terminals $S \subseteq V$, the *minimum-weight Steiner tree* problem is to find a tree in G that spans S with minimum weight. Most provable heuristics treat the network G is a metric; This assumption, in a distributed setting, cannot be easily achieved without a subtle overhead.
We give a simple distributed algorithm based on a minimum spanning tree heuristic that returns a solution whose cost is within a factor of two of the optimal. The algorithm runs in time $O(|V| \log |V|)$ on a synchronous network. We also show that another heuristic based on iteratively finding shortest paths gives a $\Theta(\log |V|)$-approximation using a novel charging scheme based on low-congestion routing on trees. Both algorithms work for unit-cost and general cost cases. The algorithms also have applications in finding multicast trees in wireless ad hoc networks.

1 Introduction

Given a network $G = (V, E)$, edge weights $w(\cdot)$, and a set of terminals $S \subseteq V$, the *minimum-weight Steiner tree* problem is to find a tree in G that spans S with minimum weight. The tree must contain all terminals; it might also contain other nodes, called *Steiner nodes*. This problem appears in various network design problems. We investigate the problem in a distributed setting.

Usually the network is treated as a distance metric over the nodes, i.e., the graph G is assumed to be a complete graph. A simple heuristic that computes a minimum spanning tree on terminals, ignoring all other nodes, gives a solution of cost within a factor of two of the optimal [1]. We call an algorithm that always returns a solution of cost within α factor of the cost of the optimal, an α-approximation algorithm. Thus, the minimum spanning tree heuristic is a 2-approximation algorithm for the Steiner tree problem. Using a centralized computation, given a network G, it is straight-forward to compute the distance

⋆ Supported by AIT graduate fellowship.
⋆⋆ Research supported by Kasetsart University Research and Development Institute (KURDI).

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 380–389, 2005.

metric and apply the heuristic. However, in distributed computation, quadratic time is required for sending information on the edges to a single node, and computing the whole metric requires also quadratic time.

In this paper, we present two algorithms for the problem. Both algorithms work for both unit-cost and general cost cases and on both synchronous and asynchronous communication network.

First, we give a simple distributed algorithm that returns a solution whose cost is within a factor of two of the optimal. The algorithm constructs the decomposition of the graph and use it to find a minimum spanning tree on the set of terminals efficiently. The algorithm runs in time $O(|V| \log |S|)$ on synchronous networks and $O(|V| \log |V|)$ on asynchronous networks.

We also show that another simple heuristic based on iteratively finding shortest paths connecting nodes to the "connected set" in any arbitrary order gives a $\Theta(\log |V|)$-approximation[1]. This iterative procedure is very similar to that used in a swarm-intelligence-based construction of mobile ad hoc multicast trees of Shen and Jaikaeo [2], if one sees the swarm agents as trying to find shortest paths. To prove the performance, we introduce a novel charging scheme based on low-congestion routing on trees. For this algorithm we concerns mainly on the performance guarantee, not the running time, because it shows that an algorithm used in practice actually does the right thing.

Although both algorithms find multicast trees for wired network, our algorithms have applications in the wireless broadcasting network as well, through the work of Wan, Călinescu, and Yi [3]. We discuss this further in Sect. 1.3.

1.1 The Model

In this paper, we consider distributed algorithms in both asynchronous and synchronous communication networks. The network can be described as a communication graph, where processors reside at the nodes and edges represent communication links. The messages sent between nodes are assumed to have bounded lengths. In *asynchronous network*, messages sent along the link arrive at an unpredictable time. In a *synchronous network*, there is a notion of rounds, that every node communicates, and the messages sent on round i are guaranteed to be delivered before round $i+1$ starts. We refer to a more complete description of the model in Lynch's book [4].

In this paper, when we discuss the *time complexity* of the algorithm, we mean the number of rounds in the synchronous network model. For the asynchronous network, the time complexity is the worst-case number of time units from the start and the completion of the algorithm, assuming the propagation delay and the inter-message delay of each link is at most one unit. This assumption is only for the performance evaluation; the algorithm must work correctly with arbitrary delays. We also consider the *communication complexity*, which is the total number of messages the algorithm sends.

[1] Every logarithm in the paper is on base 2.

Through out the paper, we denote by n the number of nodes, m the number of edges, and k the number of terminals.

1.2 Related Work

Many algorithms exist for multicast routing in wired networks. Usually the tree construction is done heuristically based on shortest paths as in CBT [5] or PIM-SM [6]. Some architecture [7] uses the provably-good heuristic such as a minimum spanning tree to approximate a cheap multicast tree. But usually the computation of the minimum spanning tree is done in a designated centralized node.

There is a long line of study on polynomial-time approximation algorithms of metric Steiner trees. The best known algorithm is by Robins and Zelikovsky [8] whose performance approaches 1.55. When the terminals are in Euclidean space and any points on the space can be used as Steiner nodes, Arora [9] gives a polynomial-time approximation scheme.

1.3 Applications to Wireless Ad Hoc Networking

Various heuristics are developed to find energy-efficient broadcast amd multicast trees in wireless network [3, 10, 11]. For example, Wieselthier, Nguyen, and Ephremides [10] give many algorithm for finding broadcast trees based on shortest paths, minimum spanning trees, and the minimum incremental power heuristic. To get a multicast tree, they prune the broadcasting tree obtained from the previous heuristics. Wan, Călinescu, and Yi [3] analyze the theoretical performance of the proposed algorithms. They give examples showing that pruning does not help, i.e., the approximation factor can be as bad as $\Omega(n)$. They also show that any trees that approximate the minimum Steiner trees within a factor of ρ, also approximate the minimum asymmetric multicast tree up to a factor of $c\rho$ where $6 \leq c \leq 12$. If the communication is bidirectional (or symmetric), the ratio is reduced to 2ρ. Wan et al. use the minimum spanning trees over the terminals to approximate the Steiner tree. They outline the algorithm which runs in time $O(nk)$ on a synchronous network. This paper improves the running time to $O(n \log k)$.

1.4 Organization

Sect. 2 describes a 2-approximation distributed algorithm for the Steiner tree problem and proves its correctness and running time. In Sect. 3, we prove that a simple intuitive heuristic has a performance guarantee of $\Theta(\log n)$ for a graph with n nodes. The proof relies on the result in Sect. 4 where the approximation algorithm for a certain on-line routing on trees is presented.

2 A 2-Approximation Algorithm

We first review a centralized algorithm that 2-approximates the Steiner trees. We are given a graph $G = (V, E)$ with edge weights $w(\cdot)$ and a set of terminals S.

Denote by OPT the optimal Steiner tree and *opt* its cost[2]. The graph together with the weights induces a shortest path distance metric d, where $d(u, v)$ is the shortest path distance from u to v in G. From metric d, we can obtain complete graph H whose nodes are terminals with d as edge weights. Takahashi and Matsuyama[1] show that the cost of the minimum spanning tree T in H is at most twice *opt*. Taking any spanning tree of union of the set of shortest paths that makes up T, we get a tree T' in G whose cost is at most the cost of T; thus, T' approximates OPT no worse than a factor of two.

We now focus on finding the minimum spanning tree on the set of terminals. Instead of finding the metric d directly, we decompose a graph and obtain another metric d' on S. The metric d' might not be the same as d, but we show that a minimum spanning tree on metric d' is also a minimum spanning tree of the terminals on the original metric d.

We define the graph decomposition in Sect. 2.1. Then, in Sect. 2.2 we describe the algorithm and proves its correctness and its running time.

2.1 Graph Decomposition

We decompose the graph into *clusters*, each associated with a terminal. We denote by $C_t \subseteq V$ the cluster containing terminal t. A cluster C_t contains nodes whose closest terminal are t, i.e., $C_t = \{u \in V | t = \arg\min_w d(u, w)\}$. If ties occur, the node belongs to the cluster whose center has the lowest id. We call t the *center* of C_t. For each node v, let $C(v)$ be the center of the cluster that contains v. An *inter-cluster edge* is an edge $e = (u, v)$ such that $C(u) \neq C(v)$. The construction of this decomposition is described later in this section.

A weighted graph G' on clusters can be defined as follows. The nodes of G' are clusters. For simplicity, we refer to a cluster C_t by its center t. For each inter-cluster edge $e = (u, v)$, we have an edge $(C(u), C(v))$ in G'. The weight of $(C(u), C(v))$ is $d(C(u), u) + w(u, v) + d(v, C(v))$, i.e., the shortest path from $C(u)$ to $C(v)$ which go through e. The graph G' induces a shortest path metric d' on the set of terminals. The metric d' can be different from the original shortest path metric d (See, for example, Fig. 1).

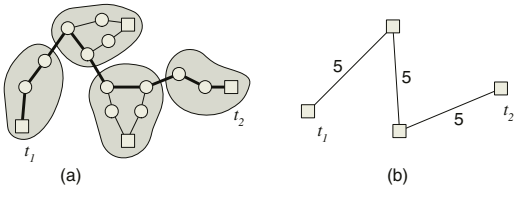


Fig. 1. (a) A decomposition of a graph G: rectangles are terminals and other circle are nodes. (b) A weighted graph G'. Note that the distance between t_1 and t_2, which is 9 in G, becomes 15 in G'.

[2] We will use the term 'weight' and 'cost' interchangably in this paper.

However, the following lemma shows that the minimum spanning trees in both metrics are the same. In the proof, we use a well-known fact (see, e.g., [12]) that for any cycle C in the graph, edge $e \in C$ having the strictly largest weight, later called a *heavy edge*, cannot be in any minimum spanning tree.

Lemma 1. *The minimum spanning tree T in G' remains a minimum spanning tree on the set of terminals in the original metric d.*

Proof. (sketch) We denote by H a weighted complete graph on the terminals with the original metric d. W.l.o.g., we assume that edge weights of H are all distinct.

First, note that from d to d', the distances only increase. Therefore, it suffices to show that if the distance for a pair (a, b) increases, i.e., $d'(a, b) > d(a, b)$, edge (a, b) cannot be in any minimum spanning tree in H, because this means that any minimum spanning trees in H have the same cost in G'.

Assume that $d'(a, b) > d(a, b)$. Consider the shortest path P in G from a to b. This path must intersect with some cluster $C_i \notin \{C_a, C_b\}$, for otherwise the distance must remain unchanged. Let $C_a = C_0, C_1, C_2, \ldots, C_k = C_b$ be a sequence of clusters that P intersects, listed in the same order as P goes from a to b. This sequence forms a path $Q = t_0, t_1, t_2, \ldots, t_k$ in H from a to b, where t_i is the center of C_i. We claim that $d(a, b) > d(t_i, t_{i+1})$ for $0 \leq i \leq k - 1$. The lemma follows from the claim because the edge (a, b) must be a heavy edge which cannot belong to any minimum spanning tree.

It is left to prove the claim. Let u be the node through which P enters C_i, i.e., u is the first node in C_i that intersects P. Also, let v be the node through which P leaves C_{i+1}, i.e., v is the last node in C_{i+1} that intersects P. Break P into three paths P_{au}, P_{uv}, and P_{vb} which are are subpaths of P from a to u, from u to v, and from v to b, respectively. Clearly, $d(a, b) = d(a, u) + d(u, v) + d(v, b)$. Since u belongs to C_{t_i}, we have[3] $d(t_i, u) < d(a, u)$. Similarly, $d(v, t_{i+1}) < d(v, b)$. Thus, $d(a, b) = d(a, u) + d(u, v) + d(v, b) > d(t_i, u) + d(u, v) + d(v, t_{i+1}) \geq d(t_i, t_{i+1})$, as required. □

It remains to show how to find the decomposition. If all edges have unit cost, i.e., $w(e) = 1$ for all $e \in E$, one can start doing parallel breadth-first search from all the terminal synchronously. Each process carries with it the id of the source terminal. Although there are many BFS processes running simultaneously, a node is visited once by the processes originated from the closest terminal. If there are ties, break ties using the terminals' id. When edges have general costs, we change from the BFS process to the Dijkstra's growing process. Both construction can be modified to work in asynchronous networks using the technique by Awerbuch [13]. The algorithm runs in asynchronous networks in time $O(|V| \log_k |V|)$ for both unit weights and general weights, using $O(k \cdot |V|^2)$ messages. Awerbuch also gives a faster algorithm for distributed shortest path computation [14]; we have not verified if this works in our case.

[3] It can be shown that the way we break ties works. We leave out the details for the simplicity of the presentation.

2.2 The Algorithm and Its Running Time

In this section, we combine the decomposition with a minimum spanning tree algorithm for asynchronous networks of Gallager, Humblet, and Spira [15] which runs in time $O(n \log n)$. This algorithm is not the best one available (for example, see [16–19]), however, we have not verified if one of these works with our decomposition.

The algorithm of Gallager *et al.* can be viewed as a parallel implementation of Boruvka's algorithm [20]. The algorithm maintains a set of connected components, initially containing every single node. For each round, each connected component finds the shortest edge connecting itself with the other component and merge. Each round takes linear time and messages, and decreases the number of components by a factor of two. Thus, the algorithm terminates in $O(\log n)$ rounds.

We follows the same steps. However, we start with clusters instead of single nodes. Also, we let the cost of the inter-cluster edges $e = (u, v)$ be $d(C(u), u) + w(u, v) + d(v, C(v))$ as in the construction of the metric d', while other edges have zero cost. Since we start with k components, the algorithm terminates in $O(\log k)$ rounds.

The running time of the approximation algorithm is the time for constructing the decomposition and the time for constructing an MST. Thus, the algorithm runs in time $O(n \log n)$.

3 A Heuristic Based on Iteratively Finding Shortest Paths

In this section we consider the following heuristic for computing Steiner trees. We index the terminals as $t_1, t_2, \ldots t_k$ and process the terminal in this order. We maintain a tree T which is initially empty. First, we connect t_1 and t_2 by the shortest path. Then for each $i > 2$, we find a shortest path from t_i to any nodes in T, and add such path which connects t_i to all terminal t_j for $j < i$. The algorithm returns T, which clearly connects all terminals. Later on, we call this heuristic a *sequence shortest-path-based heuristic*. We note that this heuristic is different from the *shortest incremental path* heuristic which guarantees to give a minimum spanning tree over the set of the terminals, as it is an implementation of Prim's algorithm.

To prove the performance guarantee for this algorithm, we consider the following on-line problem on trees. Given a tree $T = (V, E)$, we want to connect a set of terminals $S \subseteq V$ by a flow path. At each time step i, a new terminal $t_i \in V$ is revealed and we must route one unit of flow from previously connected terminals t_j, where $j < i$, to t_i along the path on the tree. We state Theorem 1, which will be proved in Sect. 4, in the form that is easy for us to use here.

Theorem 1. *For any tree T with n nodes and any sequence of nodes $t_1, t_2, \ldots t_k$ in T, we can find a set of paths $q_1, q_2, \ldots, q_{k-1}$ such that (1) q_i connects t_{i+1}*

to some t_j, such that $j \leq i$, and (2) each edge in T belongs to at most $O(\log n)$ paths.

Using this theorem, the next theorem establishes the bound on the approximation ratio of the shortest-path-based heuristic.

Theorem 2. *The sequence shortest-path-based heuristic gives a tree that spans the terminals with cost within an $O(\log n)$ factor of the optimal Steiner tree.*

Proof. (sketch) Let OPT be the optimal Steiner tree. Applying Theorem 1 on OPT with the sequence of opening nodes being $t_1, t_2, \ldots, t_k$, we get a set of $k-1$ paths $q_1, q_2, \ldots, q_{k-1}$ such that q_i connects t_{i+1} to some t_j where $j \leq i$. Theorem 1 guarantees that the sum of the cost of all paths is $O(\log n)$ of the cost of OPT, when the cost of the path q_i be the sum of the cost of its edges.

Let p_i be the path added to T in the shortest-path-based heuristic at the time the algorithm connects t_{i+1}. Clearly p_i must be no longer than any paths that connects t_{i+1} to some t_j for $j \leq i$. Thus, its cost is at most the cost of q_i, and the cost of the tree returned by the heuristic must be at most $O(\log n)$ times the optimal cost. □

We note that there is an example showing that the $O(\log n)$ approximation ratio is best possible.

4 Connecting Supplies on Trees, On-Line

Given a tree $T = (V, E)$, we want to connect a set of terminals $S \subseteq V$ by a flow path. At each time step i, a new terminal $t_i \in V$ is revealed and we must route one unit of flow from terminals t_j, where $1 \leq j < i$, to t_i along the path on the tree. At this time step i, we refer to terminals t_j, where $1 \leq j < i$ as *previously connected terminals*. Clearly, an optimal off-line algorithm would return a subtree that spans all the terminals and only route flow along these tree edges so that each edge belongs to at most one flow path. In this section, we give an on-line algorithm with the competitive ratio of $O(\log n)$ for a tree with n nodes.

The output of the algorithm for each time step i is a flow path f_i from some previously connected terminal t_j to t_i. The *amount of flow on* each edge e is the number of flow paths that use e. To prove the performance guarantee of the algorithm, it suffices to show a bound of $O(\log n)$ for the flow on any edge e.

To illustrate the idea, we first consider the case where T is a line.

4.1 Algorithm for Line Graphs

The algorithm is simple. When terminal t_i is revealed, we route the flow to the closest previously connected terminals. We assume that t_i itself is not previously connected.

For analysis simplicity, we root the graph by choose one 1-degree node to be the root. This induces the parent-child relation between nodes. We rather use

the relations *to-the-left* and *to-the-right*. We say that u is to the *left* of v when u is a predecessor of v. The *to-the-right* relationship can be defined analogously. We are ready to analyze the algorithm.

We call a pair of terminals t_a and t_b, t_a is to the left of t_b, *consecutive terminals*, if there is no other terminal t_c lying between them, i.e., there is no $t_c \neq t_a \neq t_b$ such that t_a is to the left of t_c and t_c is to the left of t_b.

The following lemma stating that each edge will be used by at most $O(\log n)$ flow paths by induction on the number of the steps. Due to space limitation, we leave out the proof.

Lemma 2. *The algorithm preserves the following property. For a pair of consecutive terminals t_a and t_b, if the distance on the line graph between them is l, the amount of flows on every edge on the path from t_a to t_b is at most $\lceil 1+log(n/l) \rceil$.*

4.2 General Case for Trees

We want to deal with a tree as a set of paths, so that we can apply the previous result on the lines. We first introduce some more definitions and notations.

Note that at any time i the set of edge with non-zero flows forms a single subtree of T; denote this subtree by T^i. To analyze the performance of the algorithm, we decompose T^i into a set of paths $P_1, P_2, \ldots, P_i$ defined recursively as $P_i = T^i - T^{i-1}$ and $P_1 = \emptyset$. Intuitively, P_i is the path that has been added at time step i. We will use the algorithm for line graphs in the previous section to route flows on these paths. P_i might be empty, because t_i is already in T^{i-1}. In this case we let P_i contains only t_i. We remind the reader that in the case that t_i is already in T^{i-1}, we still need to route a flow f_i.

We sometimes abuse the notation when we treat a subgraph as a subset of nodes. The path P_j with the smallest index j such that $j < i$ and $P_j \cap P_i \neq \emptyset$ is called a *parent path* of P_i, denoted by $par(P_i)$. If P_i is not an empty path we say that *terminal t_i is on the path P_i*, otherwise we say that *t_i is on $par(P_i)$*. The intersection node of P_i and $par(P_i)$ is called a *branching node* for P_i.

The sequence of paths $\{P_i\}$ described above can be determined solely based on the sequence of the requesting terminals. Hence, it is independent of the flow paths returned by the algorithm. Our proof analyzes the amount of flows on edge e in a fixed path P_i.

To help the routing, the algorithm maintains a set of *temporary terminals* U initially empty. A node in this set behaves like a terminal, i.e., it can send a flow to a new terminal. Each node $v \in U$ is associated with some previously connected terminal t_j. Thus, when a flow is routed from v, it actually gets routed from t_j.

We are ready to describe the algorithm. The algorithm does nothing when receiving the first terminal. When the i-th terminal is revealed it proceeds as follows. Consider the path P_i connecting t_i to T^{i-1}. If P_i is empty, i.e., t_i is already in T^{i-1}, t_i must be on some path P_j. We route the flow to t_i using the algorithm for line graphs. If P_i is not empty, consider the branching node v for P_i. If v is a terminal, we simply route the flow from v to t_i. Otherwise, we set v

to be a temporary terminal and use the line graph algorithm to route one unit of flow from terminals in $par(P_i)$ to v. We then route that flow to t_i through v. We associate with v the terminal t_i. Fig. 2 illustrates this situation.

There are two more special cases. When a temporary terminal t_i becomes a real terminals, i.e., the revealed terminal $t_i \in U$, we remove t_i from U and route one unit of flow from its associated previously connected terminal t_j. The second case is when we want to set a branching node v to be a temporary terminal for t_i but v is already a temporary terminal. In this case we route one flow from v's currently associated terminal t_j to t_i, and change the terminal associated with v to be t_i.

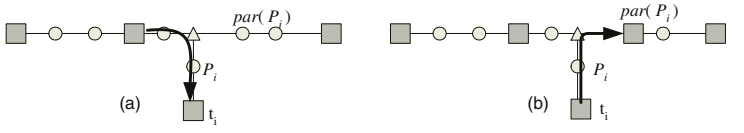


Fig. 2. (a) When t_i is revealed, one unit of flow is routed from some terminal in $par(P_i)$ through the branching node v. Then, v becomes a temporary terminal. (b) Later when some node in $par(P_i)$ is revealed and v is its closest (temporary) terminal, a flow is routed to such terminal from t_i through v.

We now prove the main result of this section.

Theorem 3. *The algorithm routes the flow paths so that no edge contains more than $O(\log n)$ amount of flows.*

Proof. On any path P_i, the amount of flows routed between terminals and temporary terminals on P_i for each edge is at most $O(\log n)$ units, from the proof for line graphs. It remains to show that the amount of flow from other path going through the branching node v of P_i is also $O(\log n)$.

We route one unit of flow when the first node t_i in P_i is revealed. After that, v acts as a temporary terminal on $par(P_i)$ and we might need to route some flow from t_i to other nodes on $par(P_i)$. However, this can happen for at most $O(\log n)$ times. To see this, note that these flows are those which go between terminals and temporary terminals on $par(P_i)$. Among these flows, there are at most $O(\log n)$ units of flow going from t_i through v, because we know that two edges adjacent to v contain at most $O(\log n)$ flows of these kind.

Finally, when v becomes real terminal or when v associates itself with the other terminal, we need to route one unit of flow from t_i, and we no longer need to route any flow from t_i through v. Thus, the total amount of flows along any edge is at most $O(\log n)$. □

Acknowledgments

We thank Chaiporn Jaikaeo for many discussions that inspire this work. We also thanks anonymous referees for their helpful comments.

References

1. Takahashi, H., Matsuyama, A.: An approximate solution for the steiner problem in graphs. Math. Jap. **24** (1980) 573–577
2. Shen, C.C., Jaikaeo, C.: Ad hoc multicast routing algorithm with swarm intelligence. Mob. Netw. Appl. **10** (2005) 47–59
3. Wan, P.J., Călinescu, G., Yi, C.W.: Minimum-power multicast routing in static ad hoc wireless networks. IEEE/ACM Trans. Netw. **12** (2004) 507–514
4. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann Publishers, Inc (1996)
5. Ballardie, T., Francis, P., Crowcroft, J.: Core based trees (CBT). In: Conference proceedings on Communications architectures, protocols and applications, ACM Press (1993) 85–95
6. Wei, L., Estrin, D.: Multicast routing in dense and sparse modes: simulation study of tradeoffs and dynamics. In: Proceedings of the 4th International Conference on Computer Communications and Networks (ICCCN '95), IEEE Computer Society (1995) 150
7. Pendarakis, D., Shi, S., Verma, D., Waldvogel, M.: ALMI: An application level multicast infrastructure. In: 3rd USNIX Symposium on Internet Technologies and Systems (USITS '01), San Francisco, CA, USA (2001) 49–60
8. Robins, G., Zelikovsky, A.: Improved steiner tree approximation in graphs. In: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2000) 770–779
9. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. J. ACM **45** (1998) 753–782
10. Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: Energy-efficient broadcast and multicast trees in wireless networks. Mob. Netw. Appl. **7** (2002) 481–492
11. Wan, P.J., Călinescu, G., Li, X.Y., Frieder, O.: Minimum-energy broadcasting in static ad hoc wireless networks. Wirel. Netw. **8** (2002) 607–617
12. Tarjan, R.E.: Data structures and network algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1983)
13. Awerbuch, B.: Complexity of network synchronization. J. ACM **32** (1985) 804–823
14. Awerbuch, B.: Randomized distributed shortest paths algorithms. In: Proceedings of the twenty-first annual ACM symposium on Theory of computing, ACM Press (1989) 490–500
15. Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. ACM Trans. Program. Lang. Syst. **5** (1983) 66–77
16. Elkin, M.: A faster distributed protocol for constructing a minimum spanning tree. In: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2004) 359–368
17. Awerbuch, B.: Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In: Proceedings of the nineteenth annual ACM conference on Theory of computing, ACM Press (1987) 230–240
18. Kutten, S., Peleg, D.: Fast distributed construction of k-dominating sets and applications. In: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing, ACM Press (1995) 238–251
19. Gafni, E.: Improvements in the time complexity of two message-optimal election algorithms. In: Proceedings of the fourth annual ACM symposium on Principles of distributed computing, ACM Press (1985) 175–185
20. Boruvka, O.: O jistém problému minimálním. Práca Morauské Přirodovědecké Společnosi **3** (1926) 37–58

A Truthful $(2-2/k)$-Approximation Mechanism for the Steiner Tree Problem with k Terminals*

Luciano Gualà[1] and Guido Proietti[1,2]

[1] Dipartimento di Informatica, Università di L'Aquila, Italy
[2] Istituto di Analisi dei Sistemi ed Informatica, CNR, Roma, Italy
{guala,proietti}@di.univaq.it

Abstract. Let a communication network be modelled by an undirected graph $G = (V, E)$ of n nodes and m edges, and assume that each edge is owned by a selfish agent, which establishes the cost of using her edge by pursuing only her personal utility. In such a non-cooperative setting, we aim at designing a *truthful mechanism* for the problem of finding a *minimum Steiner tree* of G. Since no poly-time computable exact truthful mechanism can exist for such a problem (unless P=NP), we provide a truthful $(2-2/k)$-approximation mechanism which can be computed in $O((n+k^2)m\log\alpha(m,n))$ time, where k is the number of terminal nodes, and $\alpha(\cdot,\cdot)$ is the classic inverse of the Ackermann's function. This compares favorably with the previous known $O(kn(m+n\log n))$ time and 2-approximate truthful mechanism for solving the problem.

Keywords: Steiner Tree Problem, Selfish Agents, Algorithmic Mechanism Design, Approximate Truthful Mechanisms.

1 Introduction

In any large network which contains heterogeneous components, each of the network components may be owned by different owners. Quite naturally, the incentive for an owner of a component in performing some task (e.g., forwarding a message) is to get some reward. From the network management point of view, this reward represents the price of the service of forwarding the message. Therefore, it is economically desirable that each owner declares the true price for the service that her component offers, so as to allocate the overall resources in a best possible way. Hence, it turns out that in several network applications one needs to compute efficiently a solution of a given optimization problem, with the additional constraint of enlivening the agents (through suitable payments) to cooperate with the solving algorithm. This combination of output computation and definition of payments is usually referred to as a *mechanism*.

This interplay between game theory and computational complexity is well-known by today as *algorithmic mechanism design* for selfish agents [5, 14]. Among others, in their seminal paper [14], Nisan and Ronen addressed the classic

* Work partially supported by the Research Project GRID.IT, funded by the Italian Ministry of Education, University and Research.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 390–400, 2005.

shortest path problem. This problem enjoys the fundamental property of belonging to the class of *utilitarian* problems, for which the well-known class of *Vickrey-Clarke-Groves (VCG) mechanisms* [4, 8, 19] provides an easy-to-implement corresponding truthful mechanism. Therefore, the shortest path problem can be solved efficiently. Indeed, the time complexity needed to compute the output specification and the payments to the agents, is $O(mn + n^2 \log n)$ for directed graphs, while for undirected graphs it is $O(m + n \log n)$ on a pointer machine (PM) [10, 11], and $O(m\,\alpha(m,n))$ on a word RAM [12], respectively, where $\alpha(\cdot,\cdot)$ is the classic inverse of the Ackermann's function defined in [18].

For another popular network topology, that is the *minimum spanning tree* (MST), the situation evolved similarly. Indeed, this problem naturally defines itself as utilitarian. Therefore, once again we can use a VCG-mechanism, and, as pointed out in [14], the $O(m\,\alpha(m,n))$ time and linear space sensitivity analysis algorithm in [18] can be used to solve the problem. Notice that the same time complexity holds even when each agent owns a set of edges incident to a given node [13].

Concerning another widespread network topology, that is the *single-source shortest paths tree* (SPT), it naturally admits both utilitarian and non-utilitarian formulations [9]. For the utilitarian case, in [9] the authors provide a VCG-mechanism which can be implemented in $O(mn \log \alpha(m,n))$ time on a RAM, and in $O(mn\,\alpha(m,n))$ time on a PM, while, for the non-utilitarian case, they provide a truthful mechanism (not in the VCG class) which can be implemented in $O(m + n \log n)$ time.

In this paper, we focus on another classic network design problem, that is the *Steiner Tree (ST) problem*. Let $N \subseteq V$ be a set of *terminal* nodes, with $|N| = k$. The ST problem asks for connecting all the nodes in N through a minimum weighted tree in G, possibly by making use of nodes in $V \backslash N$ (the so-called *Steiner nodes*). The only known truthful mechanism for the ST problem that we have seen to date is given in [20], where the authors provide a 2-approximate truthful mechanism, and show how to compute it in $O(kn(m + n \log n))$ time.

Here, we improve the approximation ratio to $(2 - 2/k)$, and we show that our mechanism can be efficiently computed in $O((n+k^2)m \log \alpha(m,n))$ time on a RAM, and in $O((n + k^2)m\,\alpha(m,n))$ time on a PM, by using $O(n^2)$ space in both cases. Notice that our mechanism, even on a PM, is never slower than the mechanism defined in [20], while for some value of k and m it is significantly faster. In particular, our best improvement is by a factor of $\frac{\sqrt{n}\log n}{\alpha(n,n)}$, which is obtained for $k = \Theta(\sqrt{n})$ and $m = \Theta(n)$.

2 Basic Definitions

Let $G = (V, E)$ be an undirected graph, with $|V| = n$ nodes and $|E| = m$ edges, and with a positive real weight b_e associated with each edge $e \in E$. For a given graph H other than G, we will denote its node set and its edge set by $V(H)$ and $E(H)$, respectively. Given a subgraph H of G, the *weight* of H is defined as $b(H) = \sum_{e \in E(H)} b_e$. Let T be a spanning tree of G, and let $e \in E(T)$. Let $T - e$

be the graph obtained by removing e from T, consisting of two trees with node sets V_1 and V_2, respectively. The set of edges *crossing* the cut in G induced by V_1 and V_2 is defined as

$$E(G|_{T-e}) = \{(x, y) \in E \backslash \{e\} \mid (x \in V_1) \wedge (y \in V_2)\}.$$

Let π be a path in G which passes through a pair of nodes x and y. Then, we denote by $\pi[x, y]$ the subpath of π joining x and y. A shortest path in G between two nodes r and s will be denoted by $P_G(r, s)$, while $d_G(r, s)$ will indicate the weight of such a path, also known as the *distance* in G between r and s. Finally, given a source node $r \in V$, we denote by $S(r)$ an SPT of G rooted in r.

Let a communication network be modelled by an undirected graph G, and assume that each edge is owned by a selfish agent A_e, which holds a private information t_e. We call this value the *input type* of the agent A_e. This value depends on various factors (e.g., bandwidth, reliability, etc.), and we assume that t_e represents the true cost for the agent A_e for forwarding a message through the link e. Only agent A_e knows t_e, while everything else is public knowledge. Each agent has to declare a public *bid* b_e to the mechanism. We will denote by t the vector of input types, and by b the vector of bids.

For a given optimization problem defined on G, there exists some set of feasible solutions $\mathcal{F}$ that the mechanism is allowed to choose. For each feasible solution $x \in \mathcal{F}$, some measure function $\mu(x, t)$ is defined, which depends on the true types. The mechanism tries to optimize $\mu(x, t)$, but of course it does not know t directly.

Whenever an agent A_e participates to a solution x, she incurs some cost, say $c_e(t_e, x)$, depending on her private type. While t_e is known only by the agent A_e, the cost function is public. In order to offset these costs, the mechanism provides some reward to agents participating to the computed solution, i.e., the mechanism makes a *payment* $p_e(b)$ to the agent A_e for the service provided in a solution which is computed as a function of the bid vector b.

A *mechanism* is a pair $\mathcal{M} = \langle g(b), p(b) \rangle$, where $g(b)$ is an algorithm that, given agents' bids, computes a feasible solution in $\mathcal{F}$, and $p(b)$ is a scheme which describes the payments provided to the agents. A mechanism is *exact* if the returned solution is optimal.

For each agent A_e and for each solution $g(b)$ computed by the mechanism, the *utility* function of A_e is defined as $u_e(t_e, b) = p_e(b) - c_e(t_e, g(b))$. We assume that each agent is selfish, i.e., she always attempts to maximize her utility. Let b_{-e} denote the vector of all bids besides b_e; the pair (b_{-e}, b_e) will denote the vector b. We say that *truth-telling* is a *dominant strategy* for agent A_e if bidding t_e always maximizes her utility, regardless of what the other agents bid, i.e., $u_e(t_e, (b_{-e}, t_e)) \geq u_e(t_e, (b_{-e}, b_e))$, for all b_{-e} and b_e. A mechanism is said to be *truthful* if, for every agent, truth-telling is a dominant strategy. Moreover, let $\varepsilon(\sigma)$ denote a positive real function of the input size σ. Then, an $\varepsilon(\sigma)$-*approximation mechanism* is a mechanism which returns a solution $g(b)$ which comes within a factor $\varepsilon(\sigma)$ from the optimum, i.e., $\mu(g(b), t) \leq \varepsilon(\sigma) \cdot \mu(x^*, t)$, where x^* is an

optimal solution with respect to the vector t. Finally, we say that a mechanism is *poly-time computable* if $g(\cdot)$ and $p(\cdot)$ are computable in polynomial time.

3 The Problem

For a graph $G = (V, E)$ in which there exists a bijection between the edges and the selfish agents, let $N \subseteq V$ be a set of terminal nodes which want to establish a communication. Suppose that we aim to design a routing protocol in which the terminal nodes exchange messages through a minimum-cost network. In other words, we want to design a mechanism for the ST problem. By using the notation introduced in the previous section, the problem can be formalized as follows. The set of feasible solutions $\mathcal{F}$ is the set of all the trees in G spanning N, and the measure of a solution $T \in \mathcal{F}$ (which the mechanism tries to minimize) is $\mu(T, t) = \sum_{e \in E(T)} t_e$. Such a measure function has a particular nice form, i.e., $\mu(T, t)$ is exactly the sum of the true costs of the agents participating to the solution T. This makes the problem utilitarian, and therefore solvable through a VCG-mechanism. However, for our problem such a mechanism is not poly-time computable, since it requires finding optimal solutions for the ST problem, which is known to be NP-hard even in the Euclidean or the rectilinear metric [6]. Moreover, for utilitarian problems, in [7] it is proved that VCG-mechanisms are the only exact mechanisms. In an effort to find approximate solutions, one could try to use a fast heuristic to compute a good solution, and then applying a VCG payment scheme. Unfortunately, in [15] it is exhibited a broad class of problems for which no mechanism that uses VCG payments is truthful, if its output algorithm is suboptimal, and it easy to verify that the ST problem belongs to such class. Thus, a mechanism other than VCG must be developed to guarantee truthfulness. To this aim, in this paper we provide a truthful $(2-2/k)$-approximation mechanism inspired by the results of Archer and Tardos [1].

In the rest of the paper, we call the tree selected by the mechanism *winning tree*, and we say that each edge in such a tree *wins*, while all the other edges *lose*. In [1] it is shown that in a truthful mechanism the payment scheme is entirely determined by the rule used to select the winning tree, since the payment to edge e should depend only on b_{-e} and on whether e wins or loses. Thus, for fixed b_{-e}, there must be a threshold β_e such that if A_e bids not more than β_e, then she will win, while if A_e bids above β_e she will lose. A subgraph selection rule is said *monotone* if a losing edge can never cause itself to win by raising her bid. Moreover, the truthful mechanisms that pay zero to losing edges, and pay β_e to each winning edge, correspond exactly to the monotone subgraph selection rules in which each edge can bid high enough to lose. For this reason, we restrict ourself to *2-edge connected* graphs (i.e., connected graphs which cannot be disconnected by removing a single edge), for which a bounded threshold for each edge is guaranteed.

Theorem 1 ([1]). *Let $\mathcal{M} = \langle g(b), p(b) \rangle$ be a mechanism. If $g(b)$ implements a monotone selection rule and the payment $p_e(b)$ is defined as the threshold β_e for each winning edge e, and 0 for the losing edges, then $\mathcal{M}$ is a truthful mechanism.* □

4 An Approximate Truthful Mechanism

4.1 Definition of $g(\cdot)$

In this subsection we define the algorithm of the mechanism. This is obtained by modifying the heuristic provided in [17] in order to guarantee both the monotonicity of the selection rule and the efficiency with respect to the computation of the threshold values.

Let D be the graph whose node set is N and such that for every pair of nodes $a, b \in N$, the edge (a, b) is present and has weight $w(a, b) = d_G(a, b)$. The pseudo-code of $g(\cdot)$ is given below. The algorithm computes a tree T by *expanding* a particular MST of D, say M, i.e., by replacing each edge of M with a respective shortest path in G. Note that in general the subgraph obtained may contain cycles, and so the algorithm needs to select an MST whose expansion in G is acyclic.

Let r be any terminal node. The algorithm starts with the tree T made up only by the node r, and with M as an arbitrary MST of D. Let $N(T) \subseteq N$ denote the set of terminal nodes which are currently part of T. At any step, a new terminal node b is reached, and T is correspondingly updated. More precisely, the algorithm chooses an edge $(a, b) \in E(M)$ such that $a \in N(T)$ and $b \notin N(T)$, and it tries to expand it with its corresponding shortest path $P_G(a, b)$ without forming cycles in T. If this is not possible, it modifies M by swapping the edge (a, b) with another edge $(\tilde{a}, b)$ of equal weight which admits an acyclic expansion in T, and such that $\tilde{a}$ still belongs to T. As we will see, such an alternative edge always exists. Then, it expands $(\tilde{a}, b)$ and adds the corresponding path to T. Note that since $\tilde{a}$ has already been reached, both (a, b) and $(\tilde{a}, b)$ are edges crossing the cut induced in D by $N(T)$ and $N \backslash N(T)$. Thus, since $w(a, b) = w(\tilde{a}, b)$, it follows that the new tree M which is obtained by swapping these edges is again an MST of D. The main idea of the algorithm is that an edge (a, b) admits an acyclic expansion if there exists a shortest path π joining a and b such that: (i) $\pi[a, x]$ is already in the current T, and (ii) $\pi[x, b]$ passes through no node of T (except x).

In the pseudo-code we use the following notation. Let $e' = (a', b') \in E(M)$ be an edge of M. Then, removing e' splits M into two subtrees. We denote by $N(a')$ the node set of the subtree of M containing a', and by $N(b')$ the remaining nodes. In particular, if $a' = b'$, then $N(a')$ coincides with the entire node set N. Moreover, for each node x of T, the algorithm stores two terminal nodes $a' = \text{endpoint}_1(x)$ and $b' = \text{endpoint}_2(x)$, which satisfy the following properties: (i) (a', b') belongs to M, and (ii) $P_T(a', b')$ passes through x.

Lemma 1. *The subgraph T returned by the above algorithm is an acyclic expansion of M, and M is an MST of D.*

Proof. T is built in an incremental way by adding at any step a path $\pi[x, b]$ joining a node x and a terminal node b (Line 8). Note that x is always a node of T and that by construction $\pi[x, b]$ passes through no nodes of T (other than x). It follows that T is necessarily acyclic.

Algorithm 1 Computation of $g(\cdot)$

Input: G (as weighted w.r.t. the declared bids), N;
Output: T, M;
1: compute D; $M = \mathrm{MST}(D)$;
2: choose any terminal node $r \in N$;
3: $T := (\{r\}, \emptyset)$; $N(T) := \{r\}$; $\mathrm{endpoint}_1(r) := r$; $\mathrm{endpoint}_2(r) := r$;
4: **while** $N(T) \neq N$ **do**
5: let $(a, b) \in E(M)$ be an edge such that $a \in N(T)$ and $b \notin N(T)$;
6: $\pi := P_G(a, b)$;
7: let x be the first node of T encountered along π (if we traverse π from b to a);
8: $T := T \cup \pi[x, b]$;
9: $N(T) := N(T) \cup \{b\}$;
10: $a' := \mathrm{endpoint}_1(x)$; $b' := \mathrm{endpoint}_2(x)$; $e' := (a', b') \in M$;
11: **if** $a \in N(a')$ **then** $\tilde{a} := a'$; **else** $\tilde{a} := b'$;
12: $\forall$ node w on the path $\pi[x, b]$, set $\mathrm{endpoint}_1(w) := \tilde{a}$ and $\mathrm{endpoint}_2(w) := b$;
13: **if** $\tilde{a} \neq a$ **then** $M := M \cup \{(\tilde{a}, b)\} \backslash \{(a, b)\}$;
14: **end while**
15: **return** T, M.

To prove the minimality of M and that T is an expansion of M, we observe that at any step, given the node x of T, the algorithm locates an edge (a', b') (Line 10). It easy to see that the following properties hold: (i) (a', b') is an edge of the current M, and (ii) $\pi' = P_T(a', b')$ passes through node x. Then, the algorithm selects a node $\tilde{a}$ out of a' and b' such that $a \in N(\tilde{a})$. Since initially M is an MST of D, we will show that M remains minimum during the execution of the algorithm by proving that each swap operation performed in Line 18 does not change its total weight. There are two cases: $\tilde{a} = a$ and $\tilde{a} \neq a$.

In the first case, it is easy to see that $\pi[a, x]$ possibly represents an alternative shortest path from a to x (other than the path $P_T(a, x)$ which is a shortest path too, since it is a subpath of some shortest path). Then, the algorithm expands (a, b) with the shortest path $P_T(a, x) \cup \pi[x, b]$ by adding $\pi[x, b]$ to T, and M does not change.

In the second case $(\tilde{a} \neq a)$, let $\tilde{b} \in \{a', b'\}$ be the terminal node such that $(\tilde{a}, \tilde{b}) = (a', b')$. We claim that x has the same distance from both a and $\tilde{a}$. Indeed, suppose that $d_G(a, x) < d_G(\tilde{a}, x)$; then, the path $\pi[a, x] \cup \pi'[x, \tilde{b}]$ is strictly shorter than π', and thus $w(a, \tilde{b}) < w(\tilde{a}, \tilde{b})$, which contradicts the minimality of M since $(a, \tilde{b}) \notin E(M)$ and both $(a, \tilde{b})$ and $(\tilde{a}, \tilde{b})$ are edges crossing the cut in D individuated by $N(\tilde{a})$ and $N \backslash N(\tilde{a})$. Similarly, by supposing $d_G(\tilde{a}, x) < d_G(a, x)$, we obtain that $w(\tilde{a}, b) < w(a, b)$, which is again a contradiction for the minimality of M (both $(\tilde{a}, b)$ and (a, b) are edges crossing the cut in D individuated by $N \backslash \{b\}$ and $\{b\}$). Then, the two edges (a, b) and $(\tilde{a}, b)$ have the same weight in D, and the algorithm replaces in M the edge (a, b) by the edge $(\tilde{a}, b)$, and it expands $(\tilde{a}, b)$ with the path $P_T(\tilde{a}, x) \cup \pi[x, b]$ by adding $\pi[x, b]$ to T. Clearly, the new M remains an MST, since it is obtained from the old M by swapping edges of equal weight. □

Lemma 2 ([17]). *The above algorithm is a $(2-2/k)$-approximation algorithm for the centralized ST problem.* □

Lemma 3. *The above algorithm yields a monotone selection rule.*

Proof. Let M and T be the trees returned by the algorithm, and let $e \notin E(T)$ be a losing edge. Then, e does not belong to any shortest path selected in M and thus, if A_e raises her bid, the only edges in D which increase their weight are edges in $E(D)\backslash E(M)$. Hence, the solution computed by the algorithm will be again T. □

Notice that from Lemmas 2 and 3 and from Theorem 1, we have that the above definition of $g(\cdot)$ leads to a truthful $(2-2/k)$-approximation mechanism for the ST problem, once that the payment $p_e(b)$ for each winning edge e is suitably defined as the threshold value with respect to the selection rule defined by $g(\cdot)$.

4.2 Computing the Payments

We have shown that the selection rule of our mechanism is monotone. Now, we have to compute the payments for the winning edges. We said above that the payment for a winning edge e must be equal to the threshold β_e above which the edge e loses. Thus, we have to (efficiently) compute all the threshold values.

Let $e \in E(T)$ be a winning edge. We define the *image* of e on M as

$$Im(e) = \{(a,b) \in E(M) | e \in P_T(a,b)\}.$$

Suppose that $Im(e)$ consists of a single edge of M, say (a,b). Then, as soon as b_e increases, e can exit from T for either of the following two reasons:

(i) $P_T(a,b)$ becomes longer than $P_{G-e}(a,b)$;
(ii) $P_T(a,b)$ is still a shortest path, but (a,b) is swapped with a lighter edge (a',b') in $E' = E(D|_{M-(a,b)})$ such that there exists some shortest path $P_G(a',b')$ which does not contain e.

Formally, the threshold $\beta_e^{(a,b)}$ for e with respect to an edge $(a,b) \in Im(e)$ is defined as follows: let $swap_{(a,b)}(e) = \min_{(a',b')\in E'}\{d_{G-e}(a',b')\}$; then

$$\beta_e^{(a,b)} = b_e + \min\left\{(d_{G-e}(a,b) - d_G(a,b)), (swap_{(a,b)}(e) - d_G(a,b))\right\}.$$

If $Im(e)$ consists of several edges, say $Im(e) = \{(a_1,b_1),\ldots,(a_h,b_h)\}$, clearly the threshold is

$$\beta_e = \max_{i=1,\ldots,h}\left\{\beta_e^{(a_i,b_i)}\right\}.$$

4.3 Time Complexity

Before proving our main theorem concerning the time complexity of the mechanism, we give the following lemma.

Lemma 4. *Let a, b be terminal nodes, and let $P_G(a,b)$ be a shortest path joining them. If for each $x \in V$ we are given all the distances $d_G(a,x)$ and $d_G(b,x)$, then we have that all the distances $d_{G-e}(a,b)$, for each $e \in P_G(a,b)$, can be computed in $O(m\log\alpha(m,n))$ time on a RAM, and in $O(m\,\alpha(m,n))$ time on a PM, respectively.*

Proof. Let $P_{G-e}(a,b)$ be a replacement shortest path for the edge e, i.e., a path from a to b in $G-e$ of (minimum) length $d_{G-e}(a,b)$. The problem of finding all the replacement shortest paths, one for each edge of $P_G(a,b)$, has been efficiently solved in $O(m+n\log n)$ time on a PM [11], and in $O(m\,\alpha(m,n))$ time on a word RAM [12], respectively. Both algorithms are based on a pre-computation of the SPTs $S(a)$ and $S(b)$. We now show how to improve these bounds, by using a powerful structure called *Split-Findmin* (SF) [16].

Let $e=(u,v)$ be an edge on $P_G(a,b)$, with u closer to a than v. Since a replacement shortest path $P_{G-e}(a,b)$ joining a and b must contain an edge in $E'=E(G|_{S(a)-e})$, it follows that it corresponds to a path of length

$$d_{G-e}(a,b)=\min_{f=(x,y)\in E'}\big\{k(f):=d_{G-e}(a,x)+b_f+d_{G-e}(y,b)\big\},$$

which can be shown [11] to be equivalent to

$$\begin{aligned} d_{G-e}(a,b) &= \min_{f\in E'}\big\{d_{S(a)}(a,x)+b_f+d_{S(b)}(y,b)\big\} \\ &= \min_{f\in E'}\big\{d_G(a,x)+b_f+d_G(y,b)\big\}. \end{aligned} \tag{1}$$

Hence, since we know all the distances $d_G(a,x)$ and $d_G(y,b)$ for each $x,y\in V$, $k(f)$ is available in $O(1)$ time for fixed f. It then remains to select the minimum over E'. To do this efficiently, we use an SF structure. This is a structure operating on a collection of disjoint sequences of n elements. Initially, there is only one sequence containing all the elements, and each element u has a key $k(u):=+\infty$. Then, the structure supports the following operations:

`split`(u)**:** Split the sequence containing u into two sequences of elements: one up to and including u, the other sequence containing the rest;
`findmin`(u)**:** Return the element (and the associated key) in u's sequence with minimum key;
`decrease-key`(u,k')**:** Set $k(u):=\min\{k(u),k'\}$.

We find all the distances $d_{G-e}(a,b)$ as follows. First, we label each non-tree edge f with the value (1). Then, we initialize an SF structure, where the initial n-elements sequence consists of the vertices of $S(a)$ as sorted in any arbitrary post-order. We maintain two invariants: (1) every sequence in the SF structure corresponds to some rooted subtree of $S(a)$, and (2) $k(u)$ corresponds to the label of a min-label edge connecting u to a vertex outside the u's sequence (i.e., outside the subtree of $S(a)$ currently containing u).

Let now $e=(u,v)\in P_G(a,b)$. By invariants (1) and (2), if Σ is a sequence in the SF structure and v is the root of the subtree corresponding to Σ, then `findmin`(v) will return a key $k(f_e)$, where f_e is an edge in E' belonging to $P_{G-e}(a,b)$, and $k(f_e)$ is exactly the distance $d_{G-e}(a,b)$. Once f_e is determined, we proceed to solve the problem for the child of v along the path $P_G(a,b)$, say w. Because of the post-order arrangement of the nodes, v is the rightmost element in its sequence. Then, we perform one split centered at the element preceding v in the sequence (this will sever v), and one additional split (in any arbitrary

order) for each of the children of v in $S(a)$, to reestablish invariant (1). After, we focus on the sequence associated with w, and we restore invariant (2) by performing a number of decrease-key operations. More precisely, for each edge $f = (w', y)$ such that the least common ancestor (LCA) of w' and y is v, and w' is a descendant of w in $S(a)$, we issue the operation `decrease-key`$(w', k(f))$.

Concerning the time complexity, since all the distances $d_G(a, x)$ and $d_G(y, b)$ for each $v \in V$ are given, we can compute $S(a)$ and $S(b)$ in $O(m)$ time. Moreover, since $k(f)$ is available in $O(1)$ time for a fixed non-tree edge f, it follows that labelling all the non-tree edges takes $O(m)$ time. Concerning the SF structure operations, in total there are $O(m)$ operations: $O(n)$ splits (one for each subtree whose root is adjacent to some node of $P_G(a, b)$), $O(n)$ findmins (one for each node of $P_G(a, b)$), and $O(m)$ decrease-keys (at most one for each non-tree edge). This takes $O(m \log \alpha(m, n))$ time on a RAM and $O(m\, \alpha(m, n))$ time on a PM, respectively [16]. Other costs, such as the post-order traversal and finding LCAs, are linear in both computational models [2]. □

Theorem 2. *Let $\mathcal{M}$ be the mechanism defined above. The running time to compute the winning tree T and the corresponding payments is $O((n + k^2)m \times \log \alpha(m, n))$ on a RAM, and $O((n+k^2)m\, \alpha(m, n))$ on a PM, respectively. The space used is $O(n^2)$ in both cases.*

Proof. To build D it suffices computing all the distances $d_G(a, b)$ for each pair $a, b \in N$. This can be done by solving in G the all-to-all distances problem in $O(mn \log \alpha(m, n))$ time on a RAM, and $O(mn\, \alpha(m, n))$ time on a PM [16]. Computing $M = \mathrm{MST}(D)$ takes $O(k^2 \alpha(k^2, k))$ time [3], which is equal to $O(k^2)$, since $O(\alpha(k^2, k)) = O(1)$. Moreover, building the solution T and the relating M takes $O(k(m + n))$ time, since $g(\cdot)$ is composed of $k - 1$ phases, and each phase can clearly be completed in $O(m + n)$ time.

Concerning the payment scheme, we have to compute the thresholds β_e for each winning edge $e \in E(T)$. We start by pre-computing all the distances $d_{G-e}(a, b)$, for each $(a, b) \in E(D)$ and each $e \in P_G(a, b)$. Since we have pre-computed the all-pairs distances in G, by Lemma 4 this takes $O(k^2 m \log \alpha(m, n))$ time on a RAM, and $O(k^2 m\, \alpha(m, n))$ time on a PM.

Let e be a winning edge, and let $Im(e) = \{(a_1, b_1), \ldots, (a_h, b_h)\}$. Then, to compute the threshold β_e, we have to find for every $j = 1, \ldots, h$: (i) the distances $d_{G-e}(a_j, b_j)$, and (ii) the values $swap_{(a_j,b_j)}(e)$. By the previous pre-computation, each of the former values is available in $O(1)$ time. Concerning the latter ones, we recall that the definition of this value is:

$$swap_{(a_j,b_j)}(e) = \min_{(a',b') \in E(D|_{M-(a_j,b_j)})} \{d_{G-e}(a', b')\}.$$

Once again, by the previous pre-computation, the distances $d_{G-e}(a', b')$ are known in constant time (since $d_{G-e}(a', b') = d_G(a', b')$ if $e \notin P_G(a', b')$). Thus, for each j we have to find a minimum over $O(k^2)$ values. Since $h = O(k)$, computing all the swap values could take $O(k^3)$ time by using a brute-force approach. However, we now show how to find all the $swap_{(a_j,b_j)}(e)$ values, for a

fixed winning edge e, in $O(k^2)$ time. We label each $(a',b') \in E(D)\backslash E(M)$ with the value $d_{G-e}(a',b')$. Then, for every edge (a_j,b_j) of M, we find a lightest non-tree edge which forms a cycle with (a_j,b_j), with the usual sensitivity analysis technique [18]. In this way, it is clear that each (a_j,b_j) will receive its corresponding $swap_{(a_j,b_j)}(e)$ value. Once again, this takes $O(k^2\alpha(k^2,k)) = O(k^2)$ time on a PM. Since the number of winning edges is $O(n)$, finding all the swap values takes $O(k^2n)$ time. Thus, the overall time complexity is bounded by $O((n+k^2)m\log\alpha(m,n))$ on a RAM, and by $O((n+k^2)m\alpha(m,n))$ on a PM.

Concerning the space complexity, we need to store the all-pairs distance matrix and the set of distances $d_{G-e}(a',b')$, for each $(a',b') \in E(D)$ and each winning edge e. Thus, the space used is bounded by $O(n^2+k^2n)$. However, we can improve it to $O(n^2)$ as follows. We process the non-tree edges in $E(D)\backslash E(M)$ in groups of $O(k)$ edges, by obtaining $O(k)$ groups, and then we proceed in the following way: For each edge (a',b') in the current group and for each winning edge e, we compute the distances $d_{G-e}(a',b')$; then, for each winning edge, we compute the swap values (with respect to the edges in the group). Thus, each winning edge e receives $O(k)$ swap values for each edge $(a_i,b_i) \in Im(e)$, out of which the minimum is the correct one, and therefore we need to store only $O(nk) = O(n^2)$ distances. In this way, processing each group, in order to compute the swap values, takes $O(nk\log\alpha(k,k))$ time on a RAM, and $O(nk\,\alpha(k,k))$ time on a PM, respectively. This leads to an overall time complexity bounded by $O(nk^2\log\alpha(k,k))$ on a RAM, and by $O(nk^2\alpha(k,k))$ on a PM, respectively. Since $O(n\log\alpha(k,k)) = O(m\log\alpha(m,n))$ and $O(n\,\alpha(k,k)) = O(m\,\alpha(m,n))$, it follows that both on the RAM and on the PM we get the same time complexity as when we do not group edges, and the claim follows. □

References

1. A. Archer and É. Tardos, Frugal path mechanisms, *Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'02)*, 991–999, 2002.
2. A.L. Buchsbaum, H. Kaplan, A. Rogers, and J. Westbrook, Linear-time pointer-machine algorithms for least common ancestors, MST verification, and dominators, *Proc. of the 30th Annual ACM Symposium on Theory of Computing (STOC'98)*, 279–288, 1998.
3. B. Chazelle, A minimum spanning tree algorithm with inverse-Ackermann type complexity, *J. of the ACM*, 47:1028–1047, 2000.
4. E. Clarke, Multipart pricing of public goods, *Public Choice*, 8:17–33, 1971.
5. J. Feigenbaum and S. Shenker, Incentives and Internet computation, *ACM SIGACT News*, 33(4):37–54, 2002.
6. M.R. Garey and D.S. Johnson, The rectilinear Steiner tree problem is NP-complete, *SIAM J. Appl. Math.*, 32:826–834, 1977.
7. J. Green and J.-J. Laffont, Characterization of satisfactory mechanisms for the revelation of preferences for public goods, *Econometrica*, 45(2):427–438, 1977.
8. T. Groves, Incentives in teams, *Econometrica*, 41(4):617–631, 1973.
9. L. Gualà and G. Proietti, Efficient truthful mechanisms for the single-source shortest paths tree problem, accepted for presentation at the *11th International Euro-Par Conference (EURO-PAR'05)*, Lisboa, Portugal, 2005.

10. J. Hershberger and S. Suri, Vickrey prices and shortest paths: what is an edge worth?, *Proc. of the 42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*, 252–259.
11. K. Malik, A.K. Mittal, and S.K. Gupta, The k most vital arcs in the shortest path problem, *Oper. Res. Letters*, 8:223–227, 1989.
12. E. Nardelli, G. Proietti, and P. Widmayer, A faster computation of the most vital edge of a shortest path, *Info. Proc. Letters*, 79(2):81–85, 2001.
13. E. Nardelli, G. Proietti, and P. Widmayer, Nearly linear time minimum spanning tree maintenance for transient node failures, *Algorithmica*, 40(2):119–132, 2004.
14. N. Nisan and A. Ronen, Algorithmic mechanism design, *Games and Economic Behaviour*, 35:166–196, 2001.
15. N. Nisan and A. Ronen, Computationally feasible VCG mechanisms, *Proc. of the 2nd ACM Conf. on Electronic Commerce (EC 2000)*, 242–252, 2000.
16. S. Pettie and V. Ramachandran, Computing shortest paths with comparisons and additions, *Proc. 13th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'02)*, 267–276, 2002.
17. H. Takahashi and A. Matsuyama, An approximate solution for the Steiner problem in graphs, *Math. Jap.*, 24:573–577, 1980.
18. R.E. Tarjan, Sensitivity analysis of minimum spanning trees and shortest path problems, *Inform. Proc. Lett.*, 14:30–33, 1982.
19. W. Vickrey, Counterspeculation, auctions and competitive sealed tenders, *J. of Finance*, 16:8–37, 1961.
20. W. Wang, X.-Y. Li, and Y. Wang, Truthful multicast routing in selfish wireless networks, in *Proc. of the 10th Annual Int. Conf. on Mobile Computing and Networking (INFOCOM'04)*, 245–259, 2004.

Radial Coordinate Assignment for Level Graphs*

Christian Bachmaier[1], Florian Fischer[2], and Michael Forster[3]

[1] University of Konstanz, 78457 Konstanz, Germany
christian.bachmaier@uni-konstanz.de
[2] Projective Software GmbH, 81543 Munich, Germany
florian.fischer@projective.de
[3] National ICT Australia, Eveleigh NSW 1430, Australia
michael.forster@nicta.com.au

Abstract. We present a simple linear time algorithm for drawing level graphs with a given ordering of the vertices within each level. The algorithm draws in a radial fashion without changing the vertex ordering, and therefore without introducing new edge crossings. Edges are drawn as sequences of spiral segments with at most two bends.

1 Introduction

In hierarchical graph layout, vertices are usually placed on parallel horizontal lines, and edges are drawn as polylines, which may bend when they intersect a level line. The standard drawing algorithm [9] consists of four phases: *cycle removal* (reverses appropriate edges to eliminate cycles), *level assignment* (assigns vertices to levels and introduces dummy vertices to represent edge bends), *crossing reduction* (permutes vertices on the levels), and *coordinate assignment* (assigns x-coordinates to vertices, y-coordinates are implicit through the levels).

We are especially interested in coordinate assignment. This phase is usually constrained not to change the vertex orderings computed previously. Further, it should support commonly accepted aesthetic criteria, like small area, good separation of (dummy) vertices within a level, length and slope of edges, straightness of long edges, and balancing of edges incident to the same vertex. The novelty of this paper is to draw the level lines not as parallel horizontal lines, but as concentric circles, see Fig. 1. The apparent advantage is that level graphs can be drawn with fewer edge crossings. More level graphs can be drawn without any crossing at all, i. e., planar. Radial level drawings are common, e. g., in the study of social networks [2]. There vertices model *actors* and edges represent *relations* between them. The importance (*centrality*) of an actor is expressed by closeness to the concentric center, i. e., by a low level.

2 Preliminaries

A *k-level graph* $G = (V, E, \phi)$ with $k \leq |V|$ is a graph with a level assignment $\phi\colon V \to \{1, 2, \ldots, k\}$ that partitions the vertex set into k pairwise disjoint subsets

* Part of this work was done at the University of Passau.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 401–410, 2005.

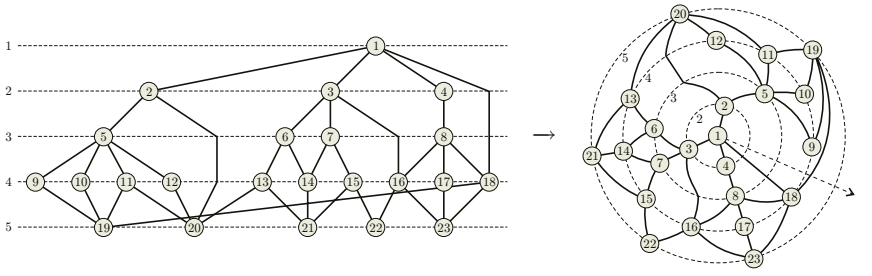


Fig. 1. Drawings of a level graph

$V = V_1 \dot\cup \cdots \dot\cup V_k$, $V_i = \phi^{-1}(i)$, $1 \le i \le k$, such that $\phi(u) \neq \phi(v)$ for each edge $(u, v) \in E$. Regardless whether G is directed or undirected, an edge incident to $u, v \in V$ is denoted by (u, v) if $\phi(u) < \phi(v)$. An edge (u, v) is *short* if $\phi(v) - \phi(u) = 1$, otherwise it is *long* and *spans* the levels $\phi(u) + 1, \ldots, \phi(v) - 1$. A level graph without long edges is *proper*. Any level graph can be made proper by subdividing each long edge $(u, v) \in E$ by the introduction of new *dummy vertices* $v_i \in B_i$, $i = \phi(u) + 1, \ldots, \phi(v) - 1$, $\phi(v_i) = i$. We draw dummy vertices as small black circles, Fig. 2(a), or edge bends, Fig. 1. The edges of a proper level graph are also called (edge) *segments*. If both end vertices of a segment are dummy vertices, it is an *inner segment*. Let $N = |V \cup B| + |E|$ denote the size of the proper level graph $G = (V \cup B, E, \phi)$ where V contains the original vertices and $B = B_1 \dot\cup \ldots \dot\cup B_k$ contains the dummy vertices with $|B| \le k|E| \le |V||E|$. For drawing level graphs it is necessary to know where long edges should be routed, i. e., between which two vertices on a spanned level. Thus we only consider proper level graphs $G = (V \cup B, E, \phi)$ in the following.

An *ordering* of a level graph is a partial order $\prec$ of $V \cup B$ such that $u \prec v$ or $v \prec u$ iff $\phi(u) = \phi(v)$. Define the (not necessarily consecutive) *positions* of the vertices as a function $\pi\colon V \cup B \to \mathbb{Z}$ with $u \prec v \Leftrightarrow \pi(u) < \pi(v)$ for any two vertices $u, v \in V_i \cup B_i$ on the same level i. In case that an ordering $\prec$ admits a level drawing of a level graph without edge crossings, $\prec$ is also called *level planar embedding*. We call an ordering of a level graph also level embedding.

3 Related Work

There are several algorithms for horizontal coordinate assignment using different approaches for the optimization of various objective functions or iterative improvement techniques, see [7] for an overview. Most interesting is the Brandes/Köpf algorithm [3], which generates at most two bends per edge and draws every inner segment vertically if no two inner segments cross. Further it minimizes the horizontal stretch of segments and also gives good results for the other aesthetic criteria. The algorithm has $\emptyset$(N) running time and is fast in practice. For level planar embeddings Eades et al. [5] presented an algorithm that does not generate bends at all. However, it may need exponential area.

3.1 Horizontal Coordinate Assignment

Since the horizontal drawing algorithm of Brandes/Köpf [3] is the basis of our algorithm, we give an extended overview. Its first two steps are carried out four times and the third step combines the results.

Vertical Alignment. The objective is to align each vertex with its left upper, right upper, left lower, and right lower median neighbor. We only describe the alignment to the left upper median, the other three passes are analogous. First, all segments are removed that do not lead to an upper median neighbor, see Fig. 2(b). Then two alignments are conflicting if their edge segments cross or share a vertex. *Type* 2 *conflicts*, two crossing inner segments, are assumed to have been avoided by the crossing reduction phase and not to occur, e. g., using the barycenter method [7]. *Type* 1 *conflicts*, a non-inner segment crossing an inner segment, are resolved in favor of the inner segment, i. e., the non-inner segment is removed from the graph. *Type* 0 *conflicts*, two crossing non-inner segments, are resolved greedily in a leftmost fashion, i. e., the right segment is removed from the graph. At this point there are no crossings left, see Fig. 2(c).

Horizontal Compaction. Each maximum set of aligned vertices, i. e., each connected component, is combined into a *block*, see Fig. 2(d). Consider the *block graph* obtained by introducing directed edges between each vertex and its successor (if any) on its level, see Fig. 2(e). A "horizontal" longest path layering[1] determines the x-coordinate of each block and thus of each contained vertex. Thereby the given minimum vertex separation δ is preserved. The block graph with expanded blocks is partitioned into *classes*, see Fig. 2(f). The first class is defined as the set of vertices which are reachable from the top left vertex. Then the class is removed from the block graph. This is repeated, until every vertex is in a class. Within the classes the graph is already compact. Now the algorithm places the classes as close as possible. In Fig. 2(f) this already happened. Fig. 2(g) shows the complete left upper layout.

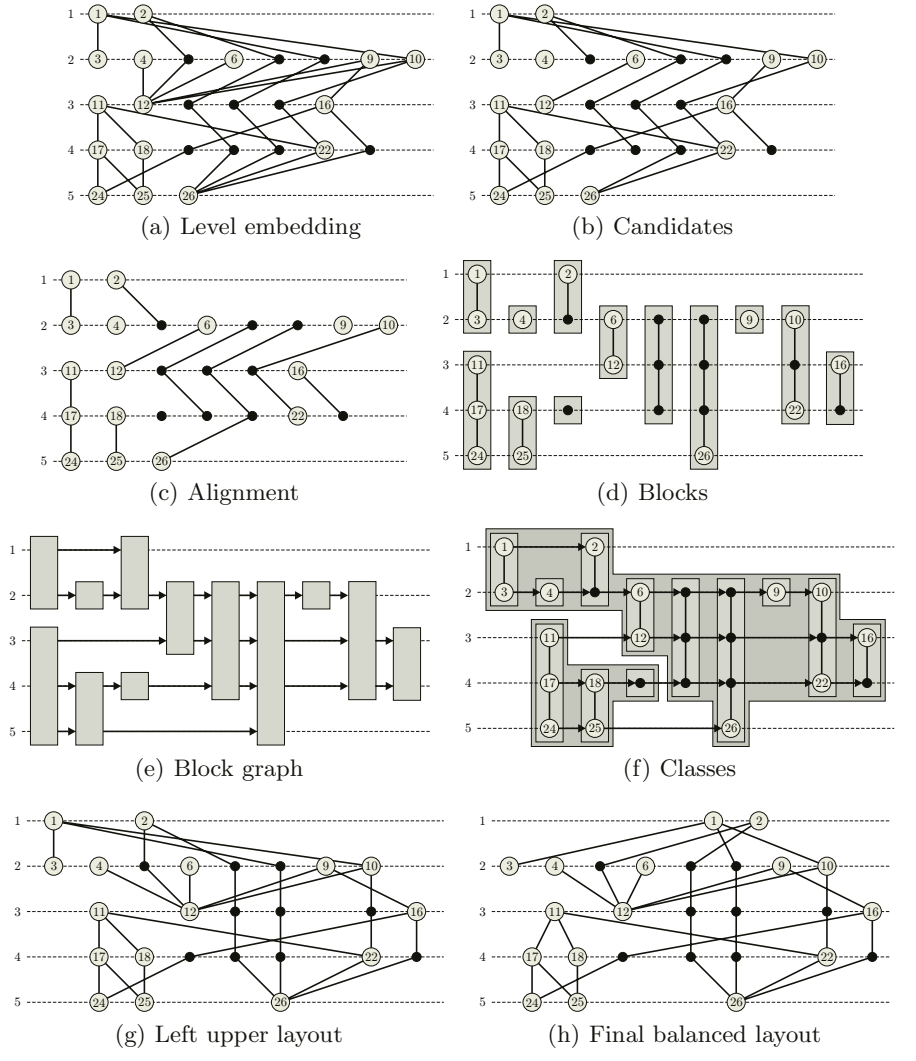
Balancing. Now each vertex has four x-coordinates. The two left (right) aligned assignments are shifted horizontally so that their minimum (maximum) coordinate agrees with the minimum (maximum) coordinate of the smallest width layout. The resulting coordinate is the average median[2] of the four intermediate coordinates. Fig. 2(h) shows the resulting drawing.

3.2 Radial Drawings

As we have already seen in Fig. 1, radial level lines help to reduce crossings. A further property of radial level lines is that they get longer with ascending level

[1] In a longest path layering vertices are assigned to levels: Each source of the graph is assigned to level 1. After removing all outgoing edges from this vertices, all (new) sources are assigned to level 2, and so on.

[2] The average median is defined as the average of the possible median values.



(a) Level embedding (b) Candidates

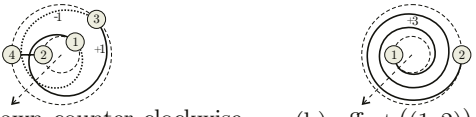
(c) Alignment (d) Blocks

(e) Block graph (f) Classes

(g) Left upper layout (h) Final balanced layout

Fig. 2. Stages of the Brandes/Köpf algorithm

number, which is especially useful if the sizes of the levels of the graph, i. e., the number of vertices, grow with the level number. The radial level idea originates from the ring diagrams of Reggiani and Marchetti [8] and from the radial tree drawings of Eades [4], which are common for free trees. There is an algorithm to compute a radial level planar embedding if one exists [1]. Thus, to maintain planarity, we require the preservation of the vertex ordering of the embedding in any case. However, our algorithm can draw arbitrary level embeddings.



(a) $(1,3)$ drawn counter clockwise and clockwise (dotted)

(b) offset $\big((1,2)\big) = +3$

Fig. 3. Offsets of edges

4 Radial Coordinate Assignment

In radial level drawings we draw the edge segments as segments of a spiral, unless they are radially aligned, in which case they are drawn as straight lines. This results in strictly monotone curves from inner to outer levels and ensures that segments do not cross inner level lines or unnecessarily each other.

For radial level embeddings it is not only necessary to know the vertex ordering, but also where the ordering starts and ends on each level. Therefore we introduce a *ray* (the dashed arrow in Fig. 1) which tags this border between the vertices. The ray is a straight halfline from the concentric center to infinity. Since the embedding maintains the position $\pi(v)$ for every vertex $v \in V \cup B$, the position of the ray is implicitly evident. We call all edges intersecting the ray *cut segments*. For a radial level embedding it is not only important to know whether an edge is a cut segment, it is also necessary to know the direction of a cut segment, clockwise or counterclockwise from inner to outer level. Otherwise the drawing is not unique, see Fig. 3(a). Further, for uniqueness it is also important to know how many times a cut segment is wound around the concentric center, i. e., how many times it crosses the ray. Since the crossing reduction phase resp. the level planar embedding algorithm mentioned in Sect. 3.2 is aware of all this information, we assume it to be given as a function $\text{offset}\colon E \to \mathbb{Z}$ with the embedding. Thereby $|\,\text{offset}(e)|$ counts the crossings of e with the ray. If $\text{offset}(e) < 0$ ($\text{offset}(e) > 0$), e is a *clockwise* (*counter clockwise*) cut segment, i. e., the sign of $\text{offset}(e)$ reflects the mathematical direction of rotation, see Fig. 3(b). If $\text{offset}(e) = 0$, e is not a cut segment and thus needs no direction information. Observe that a cut segment cannot cross the ray clockwise and counter clockwise simultaneously.

In our opinion edge bends in radial level drawings tend to be even more disturbing than in horizontal level drawings. Thus we base our algorithm on the approach of Brandes/Köpf [3] which guarantees at most two bends per edge. Further it prioritizes vertical alignment, which helps us to obtain radial alignment. The criterion of small area in horizontal coordinate assignment, i. e., to obtain small width, turns to uniform distribution of the vertices on the radial levels. As a consequence, a user parameter δ like in Sect. 3.1 is not needed.

4.1 Preprocessing

If an inner segment is a cut segment, a radial alignment of the corresponding dummy vertices cannot be guaranteed, since each inner cut segment raises the

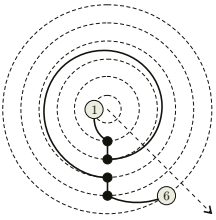


Fig. 4. Type 3 conflict

number of bends by two, see Fig. 4. We call this situation a *type* 3 *conflict*. A simple solution is to demand the absence of inner cut segments in the input embedding, similar to the type 2 conflicts. A different, more constructive and always doable approach described in the following, is to eliminate the conflicts by changing the position of the ray. This strategy changes the offset of some edges and thus changes the embedding. But this does not affect a later drawing.

Before we continue with the description of the elimination algorithm, we discuss an important property of radial level embeddings:

Lemma 1. *Let* $E'_i = \{\,(u,v) \mid u \in B_{i-1}, v \in B_i\,\} \subseteq E$ *be the set of all inner segments between levels* $i-1$ *and* i *with* $2 < i < k$*. Then for any two edges* $e_1, e_2 \in E'_i : |\operatorname{offset}(e_1) - \operatorname{offset}(e_2)| \leq 1$.

Proof. In the extreme case let $e_1, e_2 \in E'_i$ for $2 < i < k$ be inner segments with $\operatorname{offset}(e_1) = \max\{\operatorname{offset}(e) \mid e \in E'_i\}$ and $\operatorname{offset}(e_2) = \min\{\operatorname{offset}(e) \mid e \in E'_i\}$. Now assume $\operatorname{offset}(e_1) > \operatorname{offset}(e_2)+1$. As a consequence e_1 and e_2 cross. This is a type 2 conflict and contradicts the absence of type 2 conflicts in the input. □

In a first step to eliminate type 3 conflicts we consecutively *unwind* the levels in ascending order from 3 to $k-1$ with Algorithm 1. Between levels 1 and 2 resp. $k-1$ and k there are no inner segments. Clearly, level i is unwound by rotating the whole outer graph, i. e., all levels $\geq i$ are rotated by multiples of 360°. Note that UNWIND-LEVEL updates only offsets of edges between levels $i-1$ and i. The position of the ray, i. e., the ordering of the vertices, stays the same.

Algorithm 1: UNWIND-LEVEL

Input: Embedding of $G = (V \cup B, E, \phi)$ and level i with $2 < i < k$
Output: Updated offsets of inner segments entering level i

$m \leftarrow \min\{\operatorname{offset}(e) \mid e = (u,v) \in E, u \in B_{i-1}, v \in B_i\}$
foreach *segment* $e = (u,v) \in E$ *with* $v \in V_i \cup B_i$ **do** $\operatorname{offset}(e) \leftarrow \operatorname{offset}(e) - m$

Lemma 2. *After unwinding for each inner segment* $e \in E$: $\operatorname{offset}(e) \in \{0, +1\}$.

Proof. Lemma 1 implies for each inner segment $e = (u,v)$ with $\phi(v) = i$ that $\operatorname{offset}(e) \leq 1$. Additionally $\operatorname{offset}(e)$ cannot be negative since we have subtracted

the minimum over all inner segments entering level i. Since this argument holds for every level $2 < i < k$, the claim follows. □

Lemma 3. *After unwinding there are no two dummy vertices $v, v' \in B_i$ on the same level i with* offset$\big((u,v)\big) = 0$, offset$\big((u',v')\big) = +1$, *and $v \prec v'$ for any $u, u' \in B_{i-1}$.*

Proof. This follows directly from the absence of type 2 conflicts. □

Contrary to horizontal layouts we have another freedom in radial layouts without changing the crossing number: *rotation* of a single level i. A clockwise rotation, cf. Algorithm 2, is moving the vertex v with the minimum position on the ordered level $\phi(v) = i$ over the ray by setting $\pi(v)$ to a new maximum on i. A counter clockwise rotation is symmetric. Rotations do not modify the "cyclic order", i. e., the neighborhood of every vertex on its radial level line is preserved. However, the offsets of the segments incident to v must be changed. If rotating clockwise, the offsets of incoming segments of v are reduced by 1 and the offsets of outgoing segments are increased by 1. The offset updates for rotating counter clockwise are symmetric. Please note that rotation of a single level i is different to rotating levels within unwinding mentioned earlier. Here we do not rotate by (multiples of) 360° in general and do not rotate all levels $\geq i$ simultaneously.

Algorithm 2: ROTATE-CLOCKWISE

Input: Embedding of $G = (V \cup B, E, \phi)$ and level i with $1 \leq i \leq k$
Output: Updated offsets of segments entering or leaving level i

let $v = \text{argmin}\{\pi(v) \mid v \in V_i \cup B_i\}$
$\pi(v) \leftarrow \max\{\pi(v') \mid v' \in V_i \cup B_i\} + 1$
foreach *segment* $e = (u, v) \in E$ **do** offset$(e) \leftarrow$ offset$(e) - 1$
foreach *segment* $e = (v, w) \in E$ **do** offset$(e) \leftarrow$ offset$(e) + 1$

Rotation allows us to eliminate the remaining crossings of inner segments with the ray: Let $B_i' \subseteq B_i$ be the set of dummy vertices incident to an incoming inner segment $e = (u, v)$ with offset$(e) = +1$. Let $v = \text{argmax}\{\pi(v) \mid v \in B_i'\}$. We rotate level i clockwise until the ray enters the position after v, i. e., until v is the last vertex on i and thus $v = \text{argmax}\{\pi(v) \mid v \in V_i \cup B_i\}$. We use the clockwise direction, because according to Lemma 3 we do not generate new type 3 conflicts this way. Finally, all inner segments have an offset of 0. The overall running time is $\mathcal{O}(N)$.

4.2 Intermediate Horizontal Layout

In the next step we generate a horizontal layout of the radial level embedding with the Brandes/Köpf algorithm. Therefore we ignore all cut segments. Since the embedding is type 3 conflict free, all inner segments are aligned vertically.

The resulting layout will later be transformed into a concentric layout by concentrically connecting the ends of the horizontal level lines with their beginnings. Therefore, we must take into account that circumferences of radial level lines grow with ascending level numbers. Thus we use a minimum vertex separation distance $\delta_i \leftarrow \frac{1}{i}$ for each horizontal level i, which is in each case indirect proportional to i. In this way we achieve a uniform minimum arc length between two neighbor vertices on every radial level line with the radial transformation described in the next section , since we use the level numbers $1, 2, \ldots, k$ as radii.

4.3 Radial Layout

At this stage every vertex $v \in V$ has Cartesian coordinates $\big(x(v), y(v) = \phi(v)\big) \in \mathbb{R}\times\mathbb{R}$. For the transformation into a radial drawing we interpret these coordinates as polar coordinates and transform them with (1) into Cartesian coordinates $\big(x_r(v), y_r(v)\big) \in \mathbb{R} \times \mathbb{R}$. The position of the ray denotes 0°.

$$\big(x_r(v), y_r(v)\big) \leftarrow \Big(y(v) \cdot \cos\big(\tfrac{2\pi}{z} \cdot x(v)\big), y(v) \cdot \sin\big(\tfrac{2\pi}{z} \cdot x(v)\big)\Big) \tag{1}$$

The factor $\frac{2\pi}{z}$ normalizes the length of the horizontal level lines to the circumferences of the radial level lines. We set $z \leftarrow \max\big\{\max\{x(v') \mid v' \in V_i \cup B_i\} - \min\{x(v') \mid v' \in V_i \cup B_i\} + \delta_i \mid 1 \leq i \leq k\big\}$, i. e., z is the largest horizontal distance between two vertices on the same level i plus δ_i. The addend δ_i is necessary to maintain the minimum distance between the first and the last vertex, since they become neighbors on the radial level line. Let i_z be the level which defines z. The normalization automatically realizes the necessary overlap between the left and the right contour of the layout when drawn radially, see Fig. 5. Level i_z is the widest level and thus i_z defines the maximum overlap.

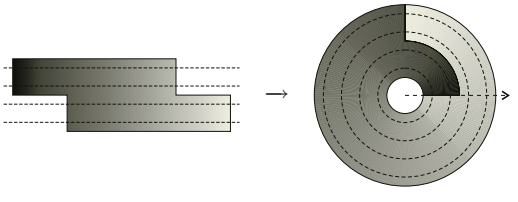


Fig. 5. Overlap of the left and right contour

After drawing the vertices, we draw the edges as segments of a spiral. Each point p of a straight line segment $e = (u, v)$ is defined by (2) for $0 \leq t \leq 1$.

$$\big(x(p), y(p)\big) = (1 - t)\big(x(u), y(u)\big) + t\big(x(v), y(v)\big) \tag{2}$$

The coordinates of p can be transformed with (1). But e can be a cut segment, which winds multiple times clockwise or counter clockwise around the center. Therefore we rather use (3) which simulates this behavior horizontally. Imagine

$|\operatorname{offset}(e)|+1$ copies of the layout placed in a row. If $\operatorname{offset}(e) \geq 0$, then imagine e drawn as straight line from u in the leftmost layout to v in the rightmost layout. Otherwise, draw e from u in the rightmost layout to v in the leftmost one. Any two neighboring layouts of the row are separated by δ_{i_z}.

$$\big(x(p), y(p)\big) = (1-t)\big(x(u), y(u)\big) + t\big(x(v) + \operatorname{offset}(e) \cdot z, y(v)\big) \tag{3}$$

For all edges with offset 0 there is only one possible direction without crossing the ray, i. e., there is only one copy in the row. Equation (3) inserted in (1) for drawing a spiral segment between u and v results in the following equation:

$$\begin{aligned}\big(x_r(p), y_r(p)\big) \leftarrow \big((1-t)y(u) + t \cdot y(v)\big) \cdot \\ \Big(\cos\Big(\tfrac{2\pi}{z} \cdot \big((1-t)x(u) + t \cdot (x(v) + \operatorname{offset}(e) \cdot z)\big)\Big), \\ \sin\Big(\tfrac{2\pi}{z} \cdot \big((1-t)x(u) + t \cdot (x(v) + \operatorname{offset}(e) \cdot z)\big)\Big)\Big)\end{aligned} \tag{4}$$

If $t = 0.5$, then p lies on a concentric circle with radius $\frac{\phi(u)+\phi(v)}{2}$, because the radius of the spiral segment grows proportional to the concentric distance between p and $\phi(u)$. To get smooth edges, the number of needed supporting points $s\colon E \to \mathbb{N}$ for drawing edges $e = (u, v)$ with an approximating polyline or spline depends on the edge length and a quality factor $q \geq 1$.

$$\begin{aligned} s(e) &\sim \phi(v) \cdot \big(\big|\tfrac{2\pi}{z} \cdot x(v) - \tfrac{2\pi}{z} \cdot x(u) + \operatorname{offset}(e) \cdot 2\pi\big|\big) \cdot q \\ &\sim \phi(v) \cdot \big(\big|\tfrac{x(v)-x(u)}{z} + \operatorname{offset}(e)\big|\big) \cdot q \end{aligned} \tag{5}$$

In the special case of $|V_1| = 1$ it is more aesthetical to place $v \in V_1$ into the concentric center, cf. Fig. 1. Thus we renumber the levels by $\phi'(w) \leftarrow \phi(w) - 1$ for all $w \in V \cup B - \{v\}$, set $x_r(v) \leftarrow y_r(v) \leftarrow 0$, layout $G' = (V \cup B - \{v\}, E - \{(v, w) \mid w \in V\}, \phi')$, and draw each edge (v, w) as straight line. In order to get a harmonic picture in the case $|V_1| > 1$, Eades [4] suggests to set the diameter of the first level to the radial distance between the radial level lines. To achieve this with our algorithm, we use $0.5, 1.5, 2.5, \ldots, k-1.5, k-0.5$ as level numbers/radii.

Usually we draw on a canvas which has dimensions $a \times b$ and has the origin in the upper left corner. Thus for each vertex or supporting point p we do the following: With the translation $\big(x_r(p), y_r(p)\big) \leftarrow \big(x_r(p) + \frac{a}{2}, y_r(p) + \frac{b}{2}\big)$ we move the origin to the center. In order to use the entire drawing space, we scale the layout by $\big(x(p), y(p)\big) \leftarrow \big(x(p), y(p)\big) \cdot \frac{\min\{a,b\}}{2k}$.

Since the elimination of type 3 conflicts generates no new crossings and (1) and (4) are bijective we do not change the crossing number given by the embedding. A radial level planar embedding is drawn planar. If we adopt the common assumption that drawing a line (here an edge as a spiral segment with its supporting points) needs $\emptyset(1)$ time, then we obtain an $\emptyset(\mathrm{N})$ running time.

5 Conclusion

We have presented a new linear time algorithm for drawing level graphs in a radial fashion. To check its performance and to visually confirm the good quality of the resulting drawings we realized a prototype as a plug-in for the Gravisto project [6] in Java. The coordinates of a graph with $N = 50,000$ can be computed in less than 50 seconds on a 1.8 GHz PC with 768 MB RAM using Java2 1.4.

Further investigations are required for radial crossing reduction algorithms that avoid type 3 conflicts already at this stage, since our elimination approach may create many crossings of non inner segments with the ray. The crossing reduction should also minimize the absolute values of the edge offsets, since this reduces crossings in general. It may be reasonable to reduce the angles spanned by the edges at the expense of a slightly increasing crossing number.

References

1. C. Bachmaier, F. J. Brandenburg, and M. Forster. Radial level planarity testing and embedding in linear time. *Journal of Graph Algorithms and Applications*, 2005. To appear. Preprint under `http://www.infosun.fmi.uni-passau.de/~chris/`.
2. U. Brandes, P. Kenis, and D. Wagner. Communicating centrality in policy network drawings. *IEEE Transact. Vis. Comput. Graph.*, 9(2):241–253, 2003.
3. U. Brandes and B. Köpf. Fast and simple horizontal coordinate assignment. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Proc. Graph Drawing, GD 2001*, volume 2265 of *LNCS*, pages 31–44. Springer, 2001.
4. P. Eades. Drawing free trees. *Bulletin of the Institute of Combinatorics and its Applications*, 5:10–36, 1992.
5. P. Eades, Q.-W. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In S. C. North, editor, *Proc. Graph Drawing, GD 1996*, volume 1190 of *LNCS*, pages 113–128. Springer, 1997.
6. Gravisto. Graph Visualization Toolkit. `http://www.gravisto.org/`. University of Passau.
7. M. Kaufmann and D. Wagner. *Drawing Graphs*, volume 2025 of *LNCS*. Springer, 2001.
8. M. G. Reggiani and F. E. Marchetti. A proposed method for representing hierarchies. *IEEE Transact. Systems, Man, and Cyb.*, 18(1):2–8, 1988.
9. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Systems, Man, and Cyb.*, 11(2):109–125, 1981.

A Theoretical Upper Bound for IP-Based Floorplanning

Guowu Yang[1], Xiaoyu Song[1], Hannah H. Yang[2], and Fei Xie[3]

[1] Dept. of ECE, Portland State University, Oregon, USA
{guowu,song}@ece.pdx.edu
[2] CAD Strategic Research Lab, Intel Corporation, Oregon, USA
hyang@ichips.intel.com
[3] Dept. Computer Science, Portland State University, Oregon, USA
xie@cs.pdx.edu

Abstract. Floorplan is a crucial estimation task in the modern layout design of systems on chips. The paper presents a novel theoretical upper bound for slicing floorplans with soft modules. We show that, given a set of soft modules of total area A_{total}, maximum area $A_{\max}$, and shape flexibility $r \geq 2.25$, there exists a slicing floorplan F of these modules such that $Area(F) \leq \min\{1.131, (1+\beta)\}A_{\text{total}}$, where $\beta = \sqrt{\frac{A_{\max}}{2rA_{\text{total}}}}$. Our results ameliorate the existing best results.

1 Introduction

With the new design paradigm of system-on-chip, systems are assembled integrating blocks of intellectual property (IP), including cores, embedded software, and bus standards as elements of systems on silicon. Due to multiple sources of components, floorplanning is an important estimation task in the modern layout design in the development of giga-transistor chips.

A floorplan dissects a rectangle into a set of non-intersecting modules, called rooms, and assigns the macro blocks into the rooms (In this paper, we only study rectangular modules, so modules are considered as rectangles). The topological relationship among the blocks is specified by the dissection. One crucial factor of most floorplanners is the representation structure which affects the effectiveness and efficiency of the optimization process directly. There are two categories of floorplans: *slicing* and *non-slicing* structures [7]. If a floorplan can be obtained by recursively cutting a rectangle into two parts by either a vertical or a horizontal line, the floorplan has a *slicing* structure. Otherwise, the floorplan has a *non-slicing* structure.

Slicing structure provides a simple way for optimizing the block orientations, defining reasonable channels in global routing and appropriately ordering the channels during detailed routing. Due to its nice properties, such as slicing tree [3], *Polish* expression [2], polynomial algorithms for special cases [4, 5], etc., it is easier to design efficient strategies to search for optimal slicing floorplans [2]. And some fast algorithms were given [10, 11]. Even if the general floorplan is not sliceable, at the initial design stage, the floorplan can be adjusted to have a slicing one by increasing space to have a fast estimation [6]. The only possible disadvantage of slicing floorplans is that even the optimal one may not pack the modules tightly [1]. Although, there are empirical evidences showing that slicing floorplans are quite good in packing modules tightly, it is important to have assurance of their performance by mathematical analysis [1].

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 411–419, 2005.

Floorplan is a challenging problem in VLSI layout design. Pan [13] proved the nonslicing floorplan is NP-complete. In the early stage of the VLSI physical design, most of the modules are not designed and so are flexible in shape, i.e., soft modules. Therefore the early reliable estimation for the areas with soft modules is helpful for the design [1, 8, 9]. In recent years, IP soft modules have been widely used in ASIC design industry [14]. In the early planning stage, not all block information is available. Some blocks are not yet designed, thus being flexible in shape. The dimension of a soft block is based on several previous implementations of the block in a library. As such, the internal logic and layout structure of a building block may have a set of alternative choices. The shape of a soft module can be changed as long as the area remains a constant and the aspect ratio is within a given range.

Recently, some excellent theoretical study [1, 8] has been performed. Young and Wong [1] proved that, given a set of modules with flexibility in shape, there exists a slicing floorplan F such that $area(F) \leq \min\{\left(1+\frac{1}{\lfloor\sqrt{r}\rfloor}\right),\ \frac{5}{4},\ (1+\alpha)\}A_{\text{total}}$, where $\alpha = \sqrt{\frac{2A_{\max}}{rA_{\text{total}}}}$, A_{total} is the total area of all the modules, $A_{\max}$ is the maximum module area, and $r \geq 2$ is the shape flexibility of each module. Note that if we do not consider the ratio of the maximum module area to the total area, the result is $area(F) \leq (1+\frac{1}{\lfloor\sqrt{r}\rfloor})A_{\text{total}}$ with $r \geq 16$, or $area(F) \leq 1.25A_{\text{total}}$ with $2 \leq r \leq 16$. Otherwise, the result is $area(F) \leq (1+\alpha)A_{\text{total}}$. Peixoto et al. [8] gave a more delicate upper bound based the same packing strategy but detailed analysis, $area(F) \leq (1+\frac{1}{\lfloor\sqrt{r}\rfloor^{\gamma-1}})A_{\text{total}}$ where γ is the smallest $j \geq 2$ such that group j (The grouping in [8] is different from ours) is not empty. When $\gamma = 1$, the upper bound in [8] is the same as in [1], and this upper bound is usable only when $r \geq 4$.

In this paper, we show that, given a set of soft rectangles of total area A_{total}, maximum area $A_{\max}$ and shape flexibility $r \geq 2.25$, there exists a slicing floorplan F of these rectangles such that $Area(F) \leq 1.131A_{\text{total}}$. The worst relative dead space of the results of [1] is 25% for $2 \leq r \leq 16$. Our result reduces their result to about 13% for $r \geq 2.25$. Moreover, our second result is stronger than the first and the third inequalities in [1] with a slightly stronger condition of $r \geq 2.1213$. Although the parameter r is slightly bigger than that in [1], it does not affect the significance of the results in practice.

2 Main Results and Related Work

We refer to the basic definitions in [1]. Let R be a rectangle. Let $height(R), width(R)$ and $area(R)$ be the height, the width and the area of R, respectively. The ratio $height(R)/width(R)$ is called as *aspect ratio of R*. A soft rectangle can have different shapes as long as the area remains the same. The shape flexibility of a soft rectangle specifies the range of its aspect ratio. A soft rectangle of area A is said to have *a shape flexibility r* if and only if R can be represented by any rectangle of area provided that:

$$\frac{1}{r} \leq \frac{height(R)}{width(R)} \leq r \tag{1}$$

Given n soft rectangles of area $A_i (i = 1, 2, ..., n)$ and a shape flexibility r, the problem is to obtain an upper bound on the area of the optimal slicing floorplan. Given a set

of soft rectangles $\Pi = \{R_1, \ldots, R_n\}, let A_i$ be the area of $R_i, i = 1, ..., n$. The total area $A_{\text{total}} = \sum A_i, A_{\text{total}} = 1$, and the maximum area $A_{\max} = \max\{A_1, \ldots, A_n\}$.

Theorem 1.1 [1]. Given a set of soft rectangles of total area A_{total}, maximum area $A_{\max}$ and shape flexibility $r \geq 2$,

(i) there exists a slicing floorplan F of the rectangles such that

$$area(F) \leq \min\left\{\left(1 + \frac{1}{\lfloor\sqrt{r}\rfloor}\right), \frac{5}{4}, (1+\alpha)\right\} A_{\text{total}} \quad \text{where} \quad \alpha = \sqrt{\frac{2A_{\max}}{rA_{\text{total}}}}$$

(ii) there exists the following relation:

$$1 \leq \frac{height(F)}{width(F)} \leq \begin{cases} \left(1 + \frac{1}{\lfloor\sqrt{r}\rfloor}\right) & area(F) \leq \left(1 + \frac{1}{\lfloor\sqrt{r}\rfloor}\right) A_{\text{total}} \\ \frac{5}{4} & area(F) \leq \frac{5}{4} A_{\text{total}} \\ \frac{1+\alpha}{(1-\frac{r\alpha}{2})^2} & area(F) \leq (1+\alpha) A_{\text{total}} \end{cases}$$

Theorem 1.2 [8]. Given a set of soft rectangles of total area A_{total}, and shape flexibility $r \geq 2$, there exists a slicing floorplan F of the rectangles such that $area(F) \leq (1 + \frac{1}{\lfloor\sqrt{r}\rfloor^{\gamma-1}}) A_{\text{total}}$ where γ is the smallest $j \geq 2$ such that group j (Where group $j = A_i | \frac{A_{\text{total}}}{r^j} \leq A_i < \frac{A_{\text{total}}}{r^{j-1}}$) is not empty. (The grouping in [1, 8] is different from ours).

We establish the following results.

Theorem 2. Given shape flexibility $r \geq 2.25$, there exists a slicing floorplan F of Π such that $Area(F) \leq 1.131 A_{\text{total}}$, and $width(F) = \sqrt{A_{\text{total}}}, \sqrt{A_{\text{total}}} \leq height(F) \leq 1.131\sqrt{A_{\text{total}}}$. Namely, the relative dead space is 13.1% .

Note that if we set $width(F) = \sqrt{A_{\text{total}}}$ and r = 2.25, the optimal upper bound of the relative dead space of a floorplan is no less than 1/9 =11.1% . For example, if there are only two modules $A_1 = 8/9, A_2 = 1/9$, the optimal relative dead space is 1/9 =11.1% .

Theorem 3. Given shape flexibility $r \geq 3\sqrt{2}/2 = 2.1213$

(i) there exists a slicing floorplan F of Π such that $area(F) \leq (1+\beta) A_{\text{total}}$ where $\beta = \sqrt{\frac{A_{\max}}{2rA_{\text{total}}}}$.

(ii) $1 \leq \frac{height(F)}{width(F)} \leq (1+\beta)\frac{A_{\text{total}}}{X^2}, X = [\frac{\sqrt{A_{\text{total}}}}{W}] \times W (\leq \sqrt{A_{\text{total}}}), W = \sqrt{\frac{rA_{\max}}{2}}$, where X is the width of the floorplan F.

Our result improves the results in [1] with a slightly stronger condition. Theorem 2 shows the relative dead space is 13.1% with $r \geq 2.25$, while the relative dead space in [1] is 25% with $2 \leq r \leq 16$. Theorem 3 shows the relative dead space β is as half as that in the third inequality in [1], namely, $\beta = 0.5\alpha$, and $\beta \leq \sqrt{\frac{A_{\max}}{2A_{\text{total}}}} \cdot \frac{1}{\lfloor\sqrt{r}\rfloor}$, with $r \geq 2.1213$. Note that α and $\frac{1}{\lfloor\sqrt{r}\rfloor}$ are the relative dead space in [1] with $r \geq 2$ and $r \geq 16$, respectively.

Remark 1: When the flexibility r is small ($2.25 \leq r < 4$), the upper bound can be chosen as the minimum of the results in Theorem 2, 3. When r is big ($r < 4$), the upper bound can be chosen as the minimum of the results in Theorem 1.2, 2, and 3.

We construct a slicing floorplan F of rectangles such that every rectangle satisfies the aspect ratio constraint (1) and the area of F is as small as possible. We will prove the main result in Section 3. Section 4 gives the conclusion. It is obvious that the amount of dead space will decrease with the flexibility and it becomes infinitely small when the rectangles have very large flexibility. We attempt to make full use of flexibility by shaping modules. Theorem 2 will be proven in Sections 3.1, 3.2, and 3.3. We prove theorem 3 in Section 3.4.

3 A Tighter Upper Bound

3.1 Classifying Modules into Groups

Given a set Π of soft rectangles, we classify rectangles into groups: G_1, G_2, ..., G_n.W.l.o.g, we assume that A_{total} =1. An area A is in G_i if and only if $\frac{1}{2^i} \leq A < \frac{1}{2^{i-1}}$ for i =1, 2, 3, ..., n. The widths of the rectangles are halved every two groups. For any module $\forall X \in G_{2i}$ and $\forall Y \in G_{2i+1}$, we have $width(X) = width(Y)$, and for $\forall X \in G_{2i-1}$ and $\forall Y \in G_{2i}$, we have $width(X)/2 = width(Y)$.

We construct a slicing floorplan Fas follows. Table 1 tabulates the areas, the heights and the widths of different groups. We pack the groups in the following order: group 1, groups 2 and 3, groups 4 and 5, group 6, and group n ($n \geq 7$). At each packing step, we place the rectangles from the largest to the smallest, and on the lowest possible level. When packing groups 2 and 3, groups 4 and 5, respectively, we always compress and align a pair of modules of the same group with the same height and the same total width (Lemma 1 and Figure 1), and then place the pairs. If the number of rectangles in group i (i= 2, 3, 4, 5) is odd, then place the single rectangle. If the number of rectangles is even in both groups 2 and 3, they can be compressed evenly in terms of lemma 1 and can be considered as group 1. Thus, we only need to consider the case where there is at most one rectangle in both groups 2 and 3, respectively. In the process of packing groups $2i$ and $2i$+1, (i= 1, 2), if the tops of the modules do not surpass the unit height over 0.125, we do not need a special shaping (just using Lemma 1). In Section 3.2, we deal with the cases that the tops of the modules surpass the unit height over 0.125. If the tops of a pair of modules (compressed evenly in terms of lemma 1) surpass the unit height over 0.125, we decompress them first and then shape them respectively. Group 6 may need shaping discussed in Section 3.3. Group n ($n \geq 7$) does not need any shaping. From the packing procedure, it is not difficult to see that the final packing gives a slicing floorplan.

There are three important aspects in our packing strategy that are different from [1]. First, we classify the module areas in a finer way: partition every group after the second group in [1] into two parts with the same width. Second, we can always compress a pair of modules of the same group aligned with the same height and the same total width (Lemma 1 and Figure 1), thus compressing modules tighter. Third, in cases 2, 3 and 4 in Section 3.2, we set the width of the left empty part as a new unit to deal with the remaining rectangles.

Let H be the height of a module, $\overline{H}$ the height of a shaped module and $\overline{w}$ the width of a shaped module.

Table 1. Classification of areas

Group	Area A	Width w	Height H
1	$1/2 \le A \le 1$	$w = 1$	$1/2 \le H \le 1$
2	$1/4 \le A < 1/2$	$w = 1/2$	$1/2 \le H < 1$
3	$1/8 \le A < 1/4$	$w = 1/2$	$1/4 \le H < 1/2$
4	$1/16 \le A < 1/8$	$w = 1/4$	$1/4 \le H < 1/2$
5	$1/32 \le A < 1/16$	$w = 1/4$	$1/8 \le H < 1/4$
6	$1/64 \le A < 1/32$	$w = 1/8$	$1/8 \le H < 1/4$
7	$1/128 \le A < 1/64$	$w = 1/8$	$1/16 \le H < 1/8$
...	.....	.....	.....

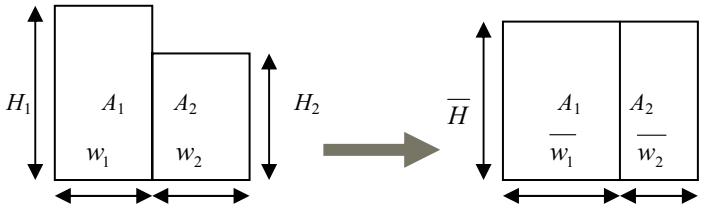


Fig. 1. Shaping in Lemma 1

Lemma 1. Any two rectangles of the same group can be aligned with the same height and the same total width.

Proof: Let $H_1(w_1)$ and $H_2(w_2)$ be the heights (widths) of the two modules, respectively (see Fig. 1). Initially, $w_1 = w_2 = w$.
(1) $\frac{1}{2} \le \frac{H_1}{w} < 1, \frac{1}{2} \le \frac{H_2}{w} < 1$. Consider the extreme case: $H_1 = w$, $H_2 = \frac{1}{2}w$. Set $\overline{H} = \frac{1}{2}(H_1 + H_2) = \frac{3}{4}w$, so we have $\overline{w_1} = \frac{4}{3}w, \overline{w_2} = \frac{2}{3}w$. We have $\overline{Hw_1} = w^2 = H_1 w$, $\overline{Hw_2} = \frac{1}{2}w^2 = H_2 w$, $\overline{w_1} + \overline{w_2} = 2w$. This means that the area of the modules remains the same, and the total width keeps the same. And we obtain $\frac{\overline{H}}{\overline{w_1}} = \frac{9}{16} \in [\frac{4}{9}, \frac{9}{4}]$, $\frac{\overline{H}}{\overline{w_2}} = \frac{9}{8} \in [\frac{4}{9}, \frac{9}{4}]$. Thus, the lemma holds.

(2) $1 \le \frac{H_1}{w} < 2, 1 \le \frac{H_2}{w} < 2$. The proof is similar to (1). □

3.2 Packing Groups 2 to 5

We only need to consider the case where there is at most one rectangle in both groups 2 and 3, respectively. The reason is discussed in 3.1. Groups n (n $\ge$ 6) are discussed in Section 3.3. In Section 3.2, we consider the following cases.

Case 1: No rectangle in either group 2 and or group 3.
Case 1.1: Even number of rectangles in both groups 4 and 5.
Case 1.2: Even number of rectangles in group 4 and odd number of rectangles in group 5.
Case 1.3: Odd number of rectangles in group 4 and even number of rectangles in group 5.
Case 1.4: Odd number of rectangle(s) in both groups 4 and 5.
Case 2: One rectangle in group 2 and no rectangle in group 3.
Case 3: No rectangle in group 2 and one rectangle A in group 3.
Case 4: One rectangle A_1 in group 2 and one rectangle A_2 in group 3.

Due to the space limitation, we only show case 1.1. The detailed algorithm can be found in [15].

Case 1: No Rectangle in Either Group 2 and or Group 3

For the rectangles in group 4 with area from 1/9 to 1/8, set $w = 1/2, 2/9 \le h < 1/4$, and pack them first. Assume the height in group 4 is from 1/4 to 4/9. For the remaining rectangles, if the number is over one, each pair of them can be compressed evenly in terms of lemma 1 and packed together.

Case 1.1: Even Number of Rectangles in Both Groups 4 and 5

If there are a pair (A_1, A_2) of rectangles whose height surpasses the unit height over 1/8, they must be in group 4 such that $A_1 : w = 1/4, H_1 = h_1 + h_1^o, h_1^o > 1/8; A_2 : w = 1/4, H_2 = h_1 + h_2^o, h_2^o \le h_1^o$; and $1/4 \le H_2 \le H_1 < 4/9, h_1 \ge h_2 \ge 0.5(h_1^o + h_2^o) > 1/8$ (See Figure 2).

Placement: Place A_1 on the right and A_2 on the left.
Shaping: Consider A_1 first
(1) If $h_1 + h_1^o > 2h_2$, the area of rectangle A_1 is bigger than the area of the empty part in the right, set: $\overline{w_1} = \frac{3}{4}\sqrt{H_1} < \frac{3}{4}\sqrt{\frac{4}{9}} = \frac{1}{2}, \overline{H_1} = \frac{4}{9}\overline{w_1} = \frac{1}{3}\sqrt{H_1} < \frac{1}{3}\sqrt{\frac{4}{9}} = \frac{2}{9}$ (We set the width of as big as possible, so lower the height). The remaining rectangles are packed on the left, so we do not need to make the width of empty part on the right that is integer times of 1/8.

Remark 2: In this case, the shadow area in Fig. 2 does not need to be used to place the remaining rectangles.

Accounting: As $h_2 > 1/8$, so we have: $\overline{H_1} - h_2 < 2/9 - 1/8 < 1/8$.
(2) If $h_1 + h_1^o \le 2h_2$, set $\overline{w_1} = 3/8, \overline{H_1} = (2/3) \cdot H_1$.

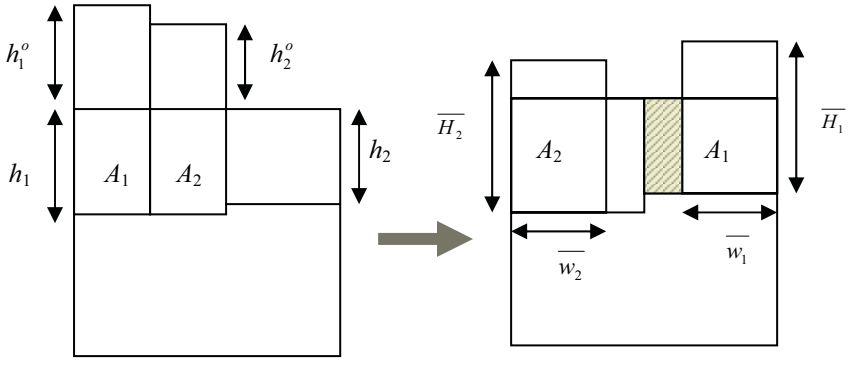


Fig. 2. Shaping in case 1.1

Remark 3: In this case, the shadow area can be used to pack the remained rectangles. The idea in the Remark 2 and 3 is used in the later cases.

Accounting: $\frac{\overline{H_1}}{\overline{w_1}} = \frac{16}{9}H_1 \geq \frac{16}{9} \cdot \frac{1}{4} = \frac{4}{9}$, and $\overline{H_1} - h_2 \leq \frac{2}{3}H_1 - \frac{1}{2}H_1 = \frac{1}{6}H_1 < \frac{2}{27} < \frac{1}{8}$.
A_2 is shaped with $\overline{w_2} = \frac{3}{8}, \overline{H_2} = \frac{2}{3}H_2$.
Accounting: $\frac{\overline{H_2}}{\overline{w_2}} = \frac{16}{9}H_2 \geq \frac{16}{9} \cdot \frac{1}{4} = \frac{4}{9}$, and $\overline{H_2} - h_1 \leq \frac{2}{3}H_2 - \frac{1}{2}H_2 = \frac{1}{6}H_2 < \frac{2}{27} < \frac{1}{8}$.

3.3 Packing Group 6 and Groups n (n $\geq$ 7)

Group 6 : $w = 1/8, 1/8 \leq H < 1/4$. We first pack group 6 without shaping. The columns with the same bottom can be exchanged freely. Since $(1/2) \cdot H < 1/8$, there are at most three columns whose tops surpass the unit height over 1/8 (totally eight columns). If there are three columns whose tops surpass the unit height over 1/8, then there are at least four columns whose tops below the unit height. First, remove the three rectangles which make the tops of the three columns surpass the unit height over 1/8. Then there are seven columns whose tops below the unit height. Therefore, there always exist three pairs of adjacent columns whose tops are below the unit height. Second, place these three rectangles on the three pairs of columns, and shape them as:

$$\overline{w} = \begin{cases} \frac{3}{2}\sqrt{A}, & \frac{1}{64} \leq A \leq \frac{1}{36} \\ \frac{1}{4}, & \frac{1}{36} \leq A \leq \frac{1}{32} \end{cases}, \overline{H} = \frac{A}{\overline{w}} \leq \frac{1}{8}$$ (Only in cases 1.2.1(b) and 1.4.1(a), we need additional height 1/8+0.006 = 13.1%).

Consider the groups $n(n \geq 7)$ with height less than 1/8. For each packing step, we leave enough space and proper width to pack the remaining rectangles. Therefore, packing the groups $n(n \geq 7)$ without any shaping can not make the tops surpass the unit height over 12.5% or 13.1% .

From the above discussion, we can draw the conclusion that, given a set of soft rectangles of total area A_{total}, maximum area $A_{\max}$, and shape flexibility $r \geq 2.25$, there exists a slicing floorplan F of these rectangles such that $Area(F) \leq 1.131 A_{\text{total}}$.

3.4 An Upper Bounds with Large Flexibility r or Small Maximum Area $A_{\max}$

In the above analysis, we do not take into account the relative sizes of the rectangles. It should be reasonable to predict a better packing if all the rectangles are small comparing with A_{total}. We consider the following strategy.

The floorplan F is initially divided into columns of width $W = \sqrt{\frac{rA_{\max}}{2}}$. We classify the areas into groups such that area A is in group i when $\frac{W^2}{4^{i-1}r} \leq A < \frac{W^2}{4^{i-2}r}$ for $i = 1, 2, 3, \ldots$ (See Table 2). An area A from group i is represented as a rectangle R of width $\frac{W}{2^{i-1}}$ and height $\frac{2^{i-1}A}{W}$. We first pack group 1, then group 2, and so on, finally group n $(n > 2)$ without any shaping. We pack the rectangles one at a time from the largest to the smallest and on the lowest possible level. The width of floorplan F is given by $X = [\frac{\sqrt{A_{\text{total}}}}{W}] \times W$, which makes the width less than or equal to the height and makes the ratio of the height to the width small. We set $H = A_{\text{total}}/X$.

Table 2. Classification of Areas in Theorem 3

Group	Area A	Width w	Height H
1	$W^2/r \le A \le A_{\max}$	$w = W$	$W/r \le H \le 2W/r$
2	$W^2/4r \le A < W^2/r$	$w = W/2$	$W/r \le H < 2W/r$
3	$W^2/16r \le A < W^2/4r$	$w = W/4$	$W/2r \le H < W/r$
4	$W^2/64r \le A < W^2/16r$	$w = W/8$	$W/4r \le H < W/2r$
…	…	…	…

We first pack the group 1: $w = W, W/r \le H \le 2W/r$. We use a similar packing technique as case 3.3. We first pack group 1 without shaping. The columns can be exchanged freely. The number of the columns whose tops surpass the height H over W/r is smaller than the total number of the columns whose tops are below the height H. By exchanging the columns, we can make the top of the adjacent column of the column whose top surpasses the height H over W/r below the height H. Shaping the rectangle as: $\overline{w} = \sqrt{rA} < 2W, \overline{H} = \overline{w}/r = \sqrt{A/r} < W/r$. The remaining rectangles are packed on the other columns.

We now pack the group 2. The Lemma 1 holds with the flexibility $r \ge 3\sqrt{2}/2 = 2.1213$. In terms of Lemma 1, we can always compress a pair of modules of group 2 every time, aligned with the same height and the same total width. If the height of the pair is over the height H by W/r, we pack the two modules on two columns respectively, and the shaping as we use for group 1.

Finally, we pack the remaining modules without any shaping. The height of the remaining is less than W/r. Through the above packing, we have:

$$\frac{Area(F)}{A_{\text{total}}} \le \frac{H + W/r}{H} = 1 + \frac{W}{rH} \le 1 + \beta, \quad \text{where} \quad \beta = \sqrt{\frac{A_{\max}}{2rA_{\text{total}}}}.$$

And we have

$$1 \le \frac{height(F)}{width(F)} \le \frac{H + W/r}{X} \le (1+\beta)\frac{A_{\text{total}}}{X^2},$$

where

$$W = \sqrt{\frac{rA_{\max}}{2}}, X = \left[\frac{\sqrt{A_{\text{total}}}}{W}\right] \times W \left(\le \sqrt{A_{\text{total}}}\right), \quad \text{and} \quad H = \frac{A_{\text{total}}}{X}.$$

4 Conclusion

We presented a theoretical upper bound for slicing floorplans with soft rectangles. We proved that, given a set of soft rectangles of total area A_{total}, maximum area $A_{\max}$, and shape flexibility $r \ge 2.25$, there exists a slicing floorplan F of these rectangles such that $Area(F) \le \min\{1.131, (1+\beta)\}A_{\text{total}}$, where $\beta = \sqrt{\frac{A_{\max}}{2rA_{\text{total}}}}$.

References

1. F.Y. Young and D.F. Wong: How good are slicing floorplans. *Integration, the VLSI Journal,* vol. 23 (1997) 61–73.
2. D.F. Wong and C. L. Liu: A new algorithm for floorplan designs. Proceedings of *the 23rd ACM/IEEE Design Automation Conference* (1986) 101–107.
3. R.H. J.M. Otten: Automatic floorplan design. Proceedings of *the 19th ACM/IEEE Design Automation Conference* (1982) 261–267.
4. G.K.H. Yeap and M. Sarrafzadeh: Sliceable floorplaning by graph dualizationn. *Siam J. Discrete Math.*, 8(2) (1995) 258–280.
5. L. Stockmeyer: Optimal orientation of cells in slicing floorplan designs. *Information and Control*, 57 (1983) 91–101.
6. M. Sarrafzadeh: Transforming an arbitrary floorplan into a sliceable one. Proceedings of*the International Conference on Computer-Aided Design* (1993) 386–389.
7. N. Sherwani: *Algorithms for VLSI physical design automation*. Kluwer Publishers (1995).
8. Helvio P. Peixoto, Margarida F. Jacome and Ander Royo: A tight area upper bound for slicing floorplans. Proceedings of13*th International conference on VLSI Design* (2000) 280–285.
9. F.Y. Young, D.F. Wong and Hannah H. Yang: Slicing Floorplans with boundary constraints. *IEEE Transactions on CADs*, vol.18, no.9 (1999) 1385–1389.
10. Weiping Shi: A fast algorithm for area minimization of slicing floorplans. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12) (1996) 1525–1532.
11. Parthasarthi Dasgupta and Susmita Sur-kolay: Slicing rectangular graphs and their optimal floorplans. *ACM Transactions on Design Automation of Electronic Systems*, vol.6, no.4, (2001) 447–470.
12. Jin Xu, Pei-Ning Giuo, and Chung-Kuan Cheng: Sequence-pair Approach for rectilinear module placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.18, no.4, (1999) 484–493.
13. P. Pan, W. Shi, and C.L. Liu: Area minimization for hierarchical floorplans. *Algorithmica*, vol.15, no.6, (1996) 550–571.
14. `http://www.eetimes.com/story/OEG20030110S0042`.
15. Guowu Yang etc: A Theoretical Upper Bound for Slicing Floorplan with Soft Modules. Technical report, Portland State University, 2004. `www.ece.pdx.edu/∼song`

Quantum Noisy Rational Function Reconstruction

Sean Hallgren[1], Alexander Russell[2], and Igor E. Shparlinski[3]

[1] NEC Laboratories America, Inc.
4 Independence Way Princeton, NJ 08540, USA
hallgren@nec-labs.com
[2] Department of Computer Science and Engineering
University of Connecticut, Storrs, CT 06269, USA
acr@cse.uconn.edu
[3] Department of Computing
Macquarie University, NSW 2109, Australia
igor@ics.mq.edu.au

Abstract. We consider the problem of determining a rational function f over a finite field $\mathbb{F}_p$ of p elements given a noisy black box $\mathcal{B}$, which for each $t \in \mathbb{F}_p$ returns several most significant bits of the residue of $f(t)$ modulo the prime p.

1 Introduction

Let p be a prime number and let $\mathbb{F}_p$ be a field of p elements. We always assume that $\mathbb{F}_p$ is represented by the elements $\{0, \ldots, p-1\}$. Given $t \in \mathbb{F}_p$, for a real $\eta \geq 0$ we denote by $\mathrm{MSB}_\eta(t)$ any $u \in \mathbb{F}_p$ such that

$$|t - u| \leq p/2^{\eta+1}. \tag{1}$$

Roughly speaking, $\mathrm{MSB}_\eta(t)$ gives η most significant bits of t; however, this definition is somewhat more flexible and better suited to our purposes. In particular we remark that η in the inequality (1) need not be an integer. One can however see a close link between $\mathrm{MSB}_\eta(t)$ and the standard notion of most significant bits.

We consider the problem of recovering a rational function $f(X) \in \mathbb{F}_p(X)$, given by a noisy black box $\mathcal{B}_\eta$ which, for each $t \in \mathbb{F}_p$, returns $\mathrm{MSB}_\eta(f(t))$ if t is not a pole of f and an arbitrary element of $\mathbb{F}_p$ if t is a pole.

It is clear that, for example, $f(X)$ and $f(X)+1$ cannot be distinguished from this kind of information, so we always assume that $f(0) = 0$.

Thus we focus on the case when f belongs to a certain class $\mathcal{F}$ of rational functions of degree at most d over $\mathbb{F}_p$ and such that for any distinct functions $f_1, f_2 \in \mathcal{F}$, $f_1 - f_2$ is not a constant function. In this case we say that $\mathcal{F}$ is *shift-free*.

For simplicity we assume that $\mathcal{F}$ is a parametric family of rational functions, although the result can be extended to more general situations.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 420–429, 2005.

Classical algorithms for some variants of the above problem have been given in [1] and [18, 19].

Throughout the paper we use $\log z$ to denote the binary logarithm of $z > 0$.

In particular, in [1] the reconstruction problem has been considered for the family of rational functions $\mathcal{F} = \{1/(X + s) \ : \ s \in \mathbb{F}_p\}$. It is shown in [1] that for any fixed $\varepsilon > 0$, if $\eta > (2/3 + \varepsilon) \log p$ then one can recover a with overwhelming probability from the values of $\mathrm{MSB}_{\eta,p}(1/(t + a))$ at polynomially many randomly chosen points $t \in \mathbb{F}_p$. More involved heuristic arguments have also been given in [1] which suggest that one can also recover a in a similar, but "noisier" scenario with $\eta > (1/3 + \varepsilon) \log p$. In Section 3.3 of [1] one can also find a conjecture that for $\eta < (1/3 - \varepsilon) \log p$ the reconstruction problem is infeasible for classical algorithms. This led the authors of [1] to suggest use of the map $t \to (t, \mathrm{MSB}_{\eta,p}(1/(t + a))$ for generating cryptographically strong pseudorandom numbers.

In [18, 19] the reconstruction problem has been considered for the family of polynomials $\mathcal{F} = \{s_d X^d + \ldots + s_1 X \in \mathbb{F}_p[X] \ : \ s_1, \ldots, s_d \in \mathbb{F}_p\}$. It is shown that in this case η can be chosen much smaller that in the case of [1], namely $\eta \ge ((d + \varepsilon) \log p)^{1/2}$ is already sufficient for a unique reconstruction $f \in \mathcal{F}$ from the values of $\mathrm{MSB}_{\eta,p}(f(t))$ at polynomially many randomly chosen points $t \in \mathbb{F}_p$.

We recall that using bit strings derived from polynomial evaluations in $\mathbb{F}_p$ has been proposed as a pseudorandom generator in [13].

Here we show strong upper bounds on the quantum query complexity of reconstructing $f \in \mathcal{F}$ for any parametric shift-free class $\mathcal{F}$ and for any $\eta > \log 6 = 2.5849 \ldots$. In particular, the suggested pseudorandom number generator of [1] may be weak against a quantum attack.

Moreover, we show that for a quantum algorithm even the value of p is not needed and in fact we design a polynomial time quantum algorithm which for any fixed $\eta > 1$ finds p from the values of $\mathrm{MSB}_{\eta,p}(f(t))$ for any (unknown) nonconstant rational function f of degree $\deg f \le p^{1/2-\varepsilon}$.

We remark that the problem studied here does not belong to the class of *hidden subgroup problems* or similar problems related to underlying group structure, whose study dates back to the celebrated Shor algorithm [16], see also [8, 12, 14, 17, 20]. Several other quantum algorithms for problems which are not explicitly associated with any underlying group structure have recently been designed in [3–5, 7, 15]. In particular, as the problem which we consider in this paper, the papers [3, 4, 15] are concerned with reconstructing a polynomial over a finite field from some partial information about its values, namely from the values $\chi(f(t))$ for a multiplicative character χ. Although here the nature of the information provided about $f(t)$ is very different, we observe that the natural amplification approach adopted in [15] yields appropriate distinction between correlated and uncorrelated states.

For a complex U and real positive V, we write $U = O(V)$ if there is an absolute constant $C > 0$ such that $|U| \le CV$ (it is more convenient for us to follow this more traditional convention rather than the one, more recently appeared

in the computer science literature, where $O(V)$ always means a nonnegative quantity).

2 Preparation

2.1 Exponential Sums

As in [15], we borrow some tools from analytic number theory, applying the *Weil bound* for exponential sums which we present in the convenient form given in [11]; see also [9, 10, 21] for more general results. Let us define

$$\mathbf{e}_p(z) = \exp(2\pi i z/p).$$

Then for any polynomials $F(X), G(X) \in \mathbb{F}_p[X]$ such that the rational function $f(X) = F(X)/G(X)$ is not constant on $\mathbb{F}_p$, the bound

$$\left| \sum_{\substack{t \in \mathbb{F}_p \\ G(t) \neq 0}} \mathbf{e}_p\left(f(t)\right) \right| \le \left(\max\{\deg F,\, \deg G\} + r - 2\right) p^{1/2} + \delta$$

holds, where

$$(r, \delta) = \begin{cases} (s, 1), & \text{if } \deg F \le \deg G, \\ (s+1, 0), & \text{if } \deg F > \deg G, \end{cases}$$

and s is the number of distinct zeros of $G(X)$. In particular

$$\left| \sum_{\substack{t \in \mathbb{F}_p \\ G(t) \neq 0}} \mathbf{e}_p\left(f(t)\right) \right| \le 2dp^{1/2} \tag{2}$$

where $d = \deg f = \max\left(\deg F, \deg G\right)$.

Using the standard technique, see [2] we immediately derive from (2) that the values of rational functions are uniformly distributed in the following sense.

Lemma 1. *For any given nonconstant rational function $f \in \mathbb{F}_p(X)$ of degree d and any integer h the number of solutions to the congruence*

$$f(t) \equiv y \pmod p, \qquad t \in \mathbb{F}_p,\ |y| \le h,$$

is $2h + O(dp^{1/2} \log p)$.

2.2 Quantum Algorithms

We refer the reader to accounts by Nielson and Chuang [14] and Kitaev, Shen and Vyalyi [8] for a discussion of quantum computation and quantum algorithms. In

particular, we use the notion of positive operator valued measurement (POVM); for example, see [17] for a discussion which matches our notation below.

We recall that a POVM P on Hilbert space $\mathcal{H}$ is a set $\mathcal{A}$ and a family $\{\vartheta_a \mid a \in \mathcal{A}\}$ of positive semidefinite operators on $\mathcal{H}$ with the property that

$$\sum_{a\in\mathcal{A}} \vartheta_a = I,$$

where I denotes the identity operator. The result of the measurement P on the state $\Psi \in \mathcal{H}$ is the probability distribution on $\mathcal{A}$ where $a \in \mathcal{A}$ is observed with probability $\langle \vartheta_a \Psi, \Psi\rangle$. Note that $\langle \vartheta_a \Psi, \Psi\rangle \geq 0$, as ϑ_α is positive semidefinite, and that

$$\sum_{a\in\mathcal{A}} \langle \vartheta_a \Psi, \Psi\rangle = \left\langle \sum_{a\in\mathcal{A}} \vartheta_a \Psi, \Psi \right\rangle = \langle I\Psi, \Psi\rangle = \|\Psi\|^2 = 1.$$

Note, also, that in the special case when $\vartheta_a = \gamma\pi$ for a projection π and a scalar $\gamma \in [0,1]$, $\langle \vartheta_a \Psi, \Psi\rangle = \gamma \|\pi\Psi\|^2$.

We remark that although $\mathrm{MSB}_{\eta,p}(x)$ is not uniquely defined we assume that the black box $\mathcal{B}_\eta$ is *consistent* and every time outputs the same value of $\mathrm{MSB}_{\eta,p}(x)$ for all integers x in the same residue class modulo p.

Hence, for any rational function $f \in \mathbb{F}_p(X)$, the sequence $\mathrm{MSB}_{\eta,p}(f(t))$, $t = 1, 2, \ldots$, is periodic with period p. Thus, it is natural to try to use a quantum period finding algorithm in order to recover p. As we have remarked the Shor algorithm [16] is not strong enough for our purposes (because the map $t \to \mathrm{MSB}_{\eta,p}(f(t))$ is not bijective). We however show that the result of [6], which we formulate below, is quite adequate for our purpose.

Given two functions Ψ and ψ which are periodic with periods L and ℓ, respectively, we denote by $D(\Psi, \psi)$ the number of integers $k \in [0, L\ell - 1]$ with $\Psi(k) \neq \psi(k)$. The following result is a version of Theorem 2 of [6].

Lemma 2. *For any given function Ψ of period L such that*

$$D(\Psi, \psi) \geq \frac{L\ell}{(\log L)^c}$$

for some constant $c > 0$ and any periodic function ψ of period $\ell < L$, there is a quantum algorithm which computes L in polynomial time with probability at least $3/4$.

3 Noisy Rational Function Reconstruction

3.1 Quantum Computational Complexity of the Field Reconstruction

In this section we concentrate on finding the unknown field size p rather than the (also unknown) function f.

Theorem 1. *Let $f \in \mathbb{F}_p(X)$ be any nonlinear rational functions of degree $d \le p^{1/2-\varepsilon}$ for some fixed ε, $1/2 > \varepsilon > 0$. Assume that we are given a consistent black box $\mathcal{B}_\eta$ which for every integer t outputs $\mathrm{MSB}_{\eta,p}(f(t))$ with some fixed $\eta \ge 1+\varepsilon$. Then there exists a quantum algorithm which in polynomial time computes p with exponentially small probability of failure.*

Proof. We show that the conditions of Lemma 2 are satisfied for the function $\Psi(t) = \mathrm{MSB}_{\eta,p}(f(t))$. Thus repeating the algorithm of Lemma 2, say, $\lceil \log p \rceil$ times we find p with probability exponentially close to 1. Let $\psi(t)$ be an arbitrary function of period $\ell < p$. Let $\mathcal{K}$ be the set of $k \in [0, p\ell - 1]$ for which simultaneously

$$\Psi(k) = \psi(k) \qquad \text{and} \qquad \Psi(k+\ell) = \psi(k+\ell).$$

Because each condition fails for at most $D(\Psi, \psi)$ values of k we obtain that

$$\#\mathcal{K} \ge p\ell - 2D(\Psi, \psi). \tag{3}$$

For each $k \in \mathcal{K}$ we have

$$\Psi(k) = \psi(k) = \psi(k+\ell) = \Psi(k+\ell),$$

thus

$$f(k) \equiv f(k+\ell) + y \pmod{p}$$

for some integer y with $|y| \le \lfloor p2^{-\eta} \rfloor$. Since the function $f(t)$ is nonlinear and $\ell < p$ we see that $f(t) - f(t+\ell)$ is a nonconstant function. Therefore Lemma 1 applies which leads to the estimate

$$\#\mathcal{K} \le \ell \left(p2^{-\eta+1} + O(dp^{1/2} \log p) \right). \tag{4}$$

Comparing (3) and (4), and recalling the conditions on η and d, we derive

$$\begin{aligned} D(\Psi, \psi) &\ge \frac{1}{2} p\ell \left(1 - 2^{-\eta+1} + O(dp^{-1/2} \log p) \right) \\ &\ge p\ell \left(\frac{1 - 2^{-\varepsilon}}{2} + O(p^{-\varepsilon} \log p) \right). \end{aligned}$$

Therefore Lemma 2 applies, which finishes the proof. □

3.2 Quantum Query Complexity of the Function Reconstruction

Here we assume that the field size p is known (for example, found by the algorithm of Section 3.1).

Theorem 2. *Let $\mathcal{F} = \{f_{\mathbf{s}} : \mathbf{s} \in \mathbb{F}_p^n\}$ be a shift-free parametric family of p^n rational functions of degree $d \le p^{1/2-\varepsilon}$ for some fixed ε, $1/2 > \varepsilon > 0$. Assume that for some $\mathbf{a} \in \mathbb{F}_p^n$ we are given a black box $\mathcal{B}_\eta$ which for every $t \in \mathbb{F}_p$ outputs $\mathrm{MSB}_{\eta,p}(f_{\mathbf{a}}(t))$ with some fixed $\eta \ge \log 6 + \varepsilon$. Then there exists a quantum algorithm which, after $O(\varepsilon^{-1}(n + \log p))$ quantum queries to $\mathcal{B}_\eta$, produces a state for which there is a POVM that determines $\mathbf{a}$ with probability at least $1 + O(p^{-1})$.*

Proof. Let us fix some k to be chosen later. For a prime p, let $\mathcal{G}$ denote a p-dimensional Hilbert space with an orthonormal basis $\{|z\rangle \mid z \in \mathbb{Z}_p\}$. Initially, by applying the Fourier transform to a δ-state, we arrive at the uniform superposition

$$\Upsilon = \frac{1}{\sqrt{p}} \sum_{t \in \mathbb{F}_p} |t\rangle \in \mathcal{G}$$

which is used to query the black-box $\mathcal{B}_\eta$. The result of the query may be computed into the phases by controlled phase shift yielding the state

$$\Psi_{\mathbf{a}} = \frac{1}{\sqrt{p}} \sum_{t \in \mathbb{F}_p} \mathbf{e}_p \left(\mathrm{MSB}_{\eta,p}(f_{\mathbf{a}}(t))\right) |t\rangle ;$$

see, for example, [8] for a discussion of quantum computation with oracles. Repeating the process independently $k \geq 1$ times yields the tensor product state

$$\Psi_{\mathbf{a},k} = \frac{1}{p^{k/2}} \sum_{\mathbf{t} \in \mathbb{F}_p^k} \mathbf{e}_p \left(\sum_{i=1}^{k} \mathrm{MSB}_{\eta,p}(f_{\mathbf{a}}(t_i)) \right) |\mathbf{t}\rangle \in \mathcal{G}^{\otimes k}$$

where $\mathbf{t} = (t_1, \ldots, t_k)$,

$$\mathcal{G}^{\otimes k} = \underbrace{\mathcal{G} \otimes \cdots \otimes \mathcal{G}}_{k},$$

and $|\mathbf{t}\rangle = |t_1\rangle \otimes \cdots \otimes |t_k\rangle$.

In general, we let $\Psi_{\mathbf{s},k}$ denote the state that would have arisen at this point had we started with the rational function $g \in \mathcal{F}$.

For any $\mathbf{s}, \mathbf{r} \in \mathbb{F}_p^n$, we have

$$\begin{aligned} |\langle \Psi_{\mathbf{s},k}, \Psi_{\mathbf{r},k} \rangle| &= \frac{1}{p^k} \left| \sum_{\mathbf{t} \in \mathbb{F}_p^k} \mathbf{e}_p \left(\sum_{i=1}^{k} \left(\mathrm{MSB}_{\eta,p}(f_{\mathbf{s}}(t_i)) - \mathrm{MSB}_{\eta,p}(f_{\mathbf{r}}(t_i)) \right) \right) \right| \\ &= \frac{1}{p^k} \prod_{i=1}^{k} \left| \sum_{t \in \mathbb{F}_p} \mathbf{e}_p \left(\mathrm{MSB}_{\eta,p}\left(f_{\mathbf{s}}(t)\right) - \mathrm{MSB}_{\eta,p}\left(f_{\mathbf{r}}(t)\right) \right) \right| . \end{aligned}$$

Observe that for $\mathbf{s} \in \mathbb{F}_p^n$, we have $\langle \Psi_{\mathbf{s},k}, \Psi_{\mathbf{s},k} \rangle = 1$. We now bound $|\langle \Psi_{\mathbf{s},k}, \Psi_{\mathbf{r},k} \rangle|$ for distinct $\mathbf{s}, \mathbf{r} \in \mathbb{F}_p^n$.

From the equation $|1 - \mathbf{e}_p(\alpha)| = 2|\sin(\pi\alpha/p)|$ we see that

$$\begin{aligned} |\,\mathbf{e}_p\left(\mathrm{MSB}_{\eta,p}\left(f_{\mathbf{s}}(t)\right) - \mathrm{MSB}_{\eta,p}\left(f_{\mathbf{r}}(t)\right)\right) - \mathbf{e}_p\left(f_{\mathbf{s}}(t) - f_{\mathbf{r}}(t)\right)| \\ \leq 2\sin\left(\pi 2^{-\eta}\right) \leq 2\sin(2^{-\varepsilon}\pi/6). \end{aligned}$$

Elementary calculus shows that $2\sin(2^{-\varepsilon}\pi/6) \leq 1 - \varepsilon$. Therefore, recalling that $\mathcal{F}$ is shift-free, we derive from (2) that

$$\begin{aligned} \left| \sum_{t \in \mathbb{F}_p} \mathbf{e}_p \left(\mathrm{MSB}_{\eta,p}(f_{\mathbf{s}}(t)) - \mathrm{MSB}_{\eta,p}(f_{\mathbf{r}}(t))\right) \right| &\leq 1 - \varepsilon + 2dp^{1/2} \\ &\leq 1 - \varepsilon + 2p^{1-\varepsilon}. \end{aligned}$$

Hence for distinct $\mathbf{s}, \mathbf{r} \in \mathbb{F}_p^n$, we have

$$|\langle \Psi_{\mathbf{s},k}, \Psi_{\mathbf{r},k} \rangle| \le \lambda^k, \tag{5}$$

where $\lambda = 1 - 0.5\varepsilon$, provided that p is large enough.

Now we show that there is a POVM that identifies the vector $\mathbf{a}$ with probability $1 + O(p^{-1})$. For each $\mathbf{s} \in \mathbb{F}_p^n$, let $\pi_{\mathbf{s},k}$ be the projection operator onto the subspace spanned by $\Psi_{\mathbf{s},k}$. As each $\pi_{\mathbf{s},k}$ is a projection operator, it is positive semidefinite, and we now show that for some $0 < \alpha < 1$ with $\alpha = 1 + O(p^{-1})$, there is a decomposition of the identity operator I of the form

$$I = \rho + \sum_{\mathbf{s} \in \mathbb{F}_p^n} \alpha \pi_{\mathbf{s},k}$$

where ρ and all $\pi_{\mathbf{s},k}$ are positive semidefinite operators on $\mathcal{G}^{\otimes k}$. Note that if $\Psi_{\mathbf{a},k}$ is measured according to this POVM, the "correct" index $(\mathbf{a}, k)$ is observed with probability α.

So define

$$\rho = I - \sum_{\mathbf{s} \in \mathbb{F}_p^n} \alpha \pi_{\mathbf{s},k}.$$

We wish to select $\alpha = 1 + O(p^{-1})$ to insure that ρ is positive semidefinite. It suffices to see that for our choice of α

$$\left\| \sum_{\mathbf{s} \in \mathbb{F}_p^n} \alpha \pi_{\mathbf{s},k} \right\| < 1, \tag{6}$$

where $\|M\|$ denotes the *operator norm* of M, given by

$$\|M\| = \sup_{\Phi \neq \mathbf{0}} \frac{\|M\Phi\|}{\|\Phi\|},$$

this supremum taken over all nonzero vectors Φ. Note that for a unit vector $\Phi \in \mathcal{G}^{\otimes k}$,

$$\sum_{\mathbf{s} \in \mathbb{F}_p^n} \alpha \pi_{\mathbf{s},k} \Phi = \sum_{\mathbf{s} \in \mathbb{F}_p^n} \langle \Phi, \Psi_{\mathbf{s},k} \rangle \Psi_{\mathbf{s},k}.$$

Let $\mathcal{E}_n$ be a Hilbert space of dimension p^n with orthonormal basis $\{\mathrm{B}_{\mathbf{s}} \mid \mathbf{s} \in \mathbb{F}_p^n\}$ and let $\tau : \mathcal{G}^{\otimes k} \to \mathcal{E}_n$ be the linear operator

$$\tau = \sum_{\mathbf{s} \in \mathbb{F}_p^n} \Psi_{\mathbf{s}} \mathrm{B}_{\mathbf{s}}^*;$$

here $\mathrm{B}_{\mathbf{s}}^* : \mathcal{E}_n \to \mathbb{C}$ is the linear functional $\mathrm{B}_{\mathbf{s}}^* : \Phi \mapsto \langle \Phi, \mathrm{B}_{\mathbf{s}} \rangle$. Then

$$\tau\tau^*(\Phi) = \sum_{\mathbf{s},\mathbf{r} \in \mathbb{F}_p^n} \Psi_{\mathbf{s}} \mathrm{B}_{\mathbf{s}}^* \left(\mathrm{B}_{\mathbf{r}} \Psi_{\mathbf{r}}^*(\Phi) \right) = \sum_{\mathbf{s} \in \mathbb{F}_p^n} \Psi_{\mathbf{s}} \Psi_{\mathbf{s}}^*(\Phi) = \sum_{\mathbf{s} \in \mathbb{F}_p^n} \pi_{\mathbf{s},k} \Phi,$$

so that $\sum_{\mathbf{s}} \pi_{\mathbf{s},k} = \tau\tau^*$; recalling that $\|\tau^*\|^2 = \|\tau\tau^*\|$, it suffices to suitably upper bound $\|\tau^*\|$. So let $\Phi \in \mathcal{G}^{\otimes k}$ be an element in the span of $\{\Psi_{\mathbf{s},k} \mid \mathbf{s} \in \mathbb{F}_p^n\}$ and let $\Gamma = \sum_{\mathbf{s}} \gamma_{\mathbf{s}} \mathrm{B}_{\mathbf{s}} \in \mathcal{E}_n$ satisfy $\tau(\Gamma) = \Phi$, which is to say that

$$\Phi = \sum_{\mathbf{s}\in\mathbb{F}_p^n} \gamma_{\mathbf{s}} \Psi_{\mathbf{s},k}.$$

Observe that

$$\begin{aligned} \|\Phi\|^2 &= \left\| \sum_{\mathbf{s}\in\mathbb{F}_p^n} \gamma_{\mathbf{s}} \Psi_{\mathbf{s},k} \right\|^2 = \sum_{\mathbf{s},\mathbf{r}\in\mathbb{F}_p^n} \gamma_{\mathbf{s}} \gamma_{\mathbf{r}}^* \langle \Psi_{\mathbf{s},k}, \Psi_{\mathbf{r},k} \rangle \\ &= \sum_{\mathbf{s}\in\mathbb{F}_p^n} |\gamma_{\mathbf{s}}|^2 + O\left(\frac{\lambda^k}{p^k} \left(\sum_{\mathbf{s}\in\mathbb{F}_p^n} |\gamma_{\mathbf{s}}| \right)^2 \right) \\ &= \|\Gamma\|^2 + O\left(\frac{\lambda^k}{p^k} \left(p^n \|\Gamma\|^2 \right) \right) = \left(1 + O\left(p^n \lambda^k\right)\right) \|\Gamma\|^2, \end{aligned} \tag{7}$$

by the Cauchy-Schwarz inequality. With Φ expressed in this way, we expand $\|\tau^*\Phi\|$ as follows:

$$\begin{aligned} \|\tau^*\Phi\|^2 &= \sum_{\mathbf{s}\in\mathbb{F}_p^n} |\langle \Phi, \Psi_{\mathbf{s},k} \rangle|^2 = \sum_{\mathbf{s}\in\mathbb{F}_p^n} \left| \left\langle \sum_{\mathbf{r}\in\mathbb{F}_p^n} \gamma_{\mathbf{r}} \Psi_{\mathbf{r},k}, \Psi_{\mathbf{s},k} \right\rangle \right|^2 \\ &= \sum_{\mathbf{s}\in\mathbb{F}_p^n} \left| \gamma_{\mathbf{s}} + \sum_{h\neq g} \gamma_{\mathbf{r}} \langle \Psi_{\mathbf{r},k}, \Psi_{\mathbf{s},k} \rangle \right|^2. \end{aligned} \tag{8}$$

Recalling the inner product bounds of (5), for any $\mathbf{s} \in \mathbb{F}_p^n$ we must have

$$\left| \sum_{\substack{\mathbf{r}\in\mathbb{F}_p^n \\ \mathbf{s}\neq\mathbf{r}}} \gamma_{\mathbf{r}} \langle \Psi_{\mathbf{r},k}, \Psi_{\mathbf{s},k} \rangle \right| \leq \lambda^k p^{-k} \sum_{\mathbf{r}\in\mathbb{F}_p^n} |\gamma_{\mathbf{r}}| \leq \lambda^k p^{-k} \sqrt{p^n} \|\Gamma\|, \tag{9}$$

again by the Cauchy-Schwarz inequality. Finally, considering that $\|\alpha + \beta\| \leq \|\alpha\| + \|\beta\|$, we conclude from (8) and (9) that

$$\|\tau^*\Phi\| \leq \|\Gamma\| + p^n \lambda^k \|\Gamma\| = \left(1 + p^n \lambda^k\right) \|\Gamma\|$$

and, from (7), that

$$\|\tau^*\Phi\| \leq \left(1 + O\left(p^n \lambda^k\right)\right) \|\Phi\|.$$

Hence

$$\left\| \sum_{\mathbf{s}\in\mathbb{F}_p^n} \pi_{\mathbf{s},k} \right\| \leq 1 + O\left(p^n \lambda^k\right).$$

Then taking $k = \lceil 2(n+1)\varepsilon^{-1} \log p \rceil$ we obtain

$$p^n \lambda^k \le p^n (1 - 0.5\varepsilon)^k \le p^n \exp(-0.5\varepsilon k) \le p^{-1}.$$

Hence, we obtain

$$\left\| \sum_{\mathbf{s} \in \mathbb{F}_p^n} \pi_{\mathbf{s},k} \right\| \le 1 + O\left(p^{-1}\right),$$

and are guaranteed that (6) holds (provided that p is large enough) for some $\alpha = 1 + O(p^{-1})$ (recall that $(1+\delta)^{-1} = 1 + O(\delta)$). Thus the above POVM determines $\mathbf{s}$ with probability $\alpha = 1 + O(p^{-1})$. □

4 Remarks and Open Questions

Simlar results can be obtained for the truncated exponential map

$$t \to \left(t, \mathrm{MSB}_{\eta,p}(a\vartheta^t)\right)$$

(if the order of ϑ modulo p is large enough) and for several more similar maps.

One can also extend our results to composite moduli. There are several additional complications in this case and, although there is little doubt that the same approach works, the final results can be weaker than those of this paper.

Probably the most important (and apparently hard) open question is to design analogues of our algorithms for a weaker black-box $\widetilde{\mathcal{B}}_\eta$ and let it accept only the integer values t of the same bits length as the unknown modulus.

References

1. D. Boneh, S. Halevi and N. A. Howgrave-Graham, 'The modular inversion hidden number problem', *Proc. Asiacrypt 2001* Queensland, Australia, Lect. Notes in Comp. Sci. Vol. 2248, Springer-Verlag, Berlin, 2001, 36–51.
2. J. H. H. Chalk, 'Polynomial congruences over incomplete residue systems modulo k', *Proc. Kon. Ned. Acad. Wetensch.*, **A92** (1989), 49–62.
3. W. van Dam, 'Quantum algorithms for weighing matrices and quadratic residues', *Algorithmica*, **34** (2002), 413–428.
4. W. van Dam, S. Hallgren and L. Ip, 'Quantum algorithms for hidden coset problems', *Proc. 14th ACM-SIAM Symp. on Discr. Algorithms*, SIAM, 2003, 489–498.
5. D. Grigoriev, 'Testing shift-equivalence of polynomials by deterministic, probabilistic and quantum machines', *Theor. Comp. Sci.*, **180** (1997), 217–228.
6. L. Hales and S. Hallgren, 'An improved quantum Fourier transform algorithm and applications', *Proc 41th IEEE Symp. on Found. of Comp. Sci.*, IEEE, 2000, 515–525.
7. S. Hallgren, 'Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem', *Proc. 34th ACM Symp. on Theory of Comp.*, ACM, 2002, 653–658.
8. A. Yu. Kitaev, A. H. Shen and M. N. Vyalyi, *Classical and quantum computation*, Graduate Studies in Mathematics, Vol.47, Amer. Math. Soc., 2002.

9. W.-C. W. Li, *Number theory with qpplications*, World Scientific, Singapore, 1996.
10. R. Lidl and H. Niederreiter, *Finite Fields*, Cambridge University Press, Cambridge, 1997.
11. C. J. Moreno and O. Moreno, 'Exponential sums and Goppa codes, 1', *Proc. Amer. Math. Soc.*, **111** (1991), 523–531.
12. M. Mosca and A. Ekert, 'The hidden subgroup problem and eigenvalue estimation on a quantum computer', *Proc. 1st NASA Intern. Conf. on Quantum Computing and Quantum Communication*, Palm Springs, USA, Lect. Notes in Comp. Sci. Vol. 1509, Springer-Verlag, Berlin, 1999, 174–188.
13. H. Niederreiter and C. P. Schnorr, 'Local randomness in polynomial random number and random function generators', *SIAM J. Comp.*, **13** (1993), 684–694.
14. M. Nielsen and I. Chuang, *Quantum computation and quantum information*, Cambridge University Press, Cambridge, 2002.
15. A. C. Russell and I. E. Shparlinski, 'Classical and quantum algorithms for function reconstruction via character evaluation', *J. Compl.*, **20** (2004), 404–422.
16. P. Shor, 'Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer', *SIAM J. Comp.*, **26** (1997), 1484–1509.
17. P. Shor, 'Quantum information theory: Results and open problems', *Geometric and Functional Analysis*, **2** (2000), 816–838.
18. I. E. Shparlinski, 'Sparse polynomial approximation in finite fields', *Proc. 33rd ACM Symp. on Theory of Comput.*, Crete, Greece, July 6-8, (2001), 209–215.
19. I. E. Shparlinski and A. Winterhof, 'Noisy interpolation of sparse polynomials in finite fields', *Appl. Algebra in Engin., Commun. and Computing*, (to appear).
20. D. R. Simon, 'On the power of quantum computation', *SIAM J. Comp.*, **26** (1997), 1474–1483.
21. A. Weil, *Basic number theory*, Springer-Verlag, New York, 1974.

Promised and Distributed Quantum Search*

Shengyu Zhang

Computer Science Department, Princeton University, NJ 08544, USA
szhang@cs.princeton.edu

Abstract. This paper gives a quantum algorithm to search in an set S for a k-tuple satisfying some predefined relation, with the promise that some components of a desired k-tuple are in some subsets of S. In particular when $k = 2$, we show a tight bound of the quantum query complexity for the Claw Finding problem, improving previous upper and lower bounds by Buhrman, Durr, Heiligman, Hoyer, Magniez, Santha and de Wolf [7].

We also consider the distributed scenario, where two parties each holds an n-element set, and they want to decide whether the two sets share a common element. We show a family of protocols *s.t.* $q(P)^{3/2} \cdot c(P) = O(n^2 \log n)$, where $q(P)$ and $c(P)$ are the number of quantum queries and the number of communication qubits that the protocol P makes, respectively. This implies that we can pay more for quantum queries to save on quantum communication, and vice versa. To our knowledge, it is the first result about the tradeoff between the two resources.

1 Introduction

Recently Ambainis [5] proposed a novel algorithm for k-Element Distinctness, which is to decide whether there are k equal elements in a given set A of size N. As later pointed out by Magniez, Santha and Szegedy in [13] and by Childs and Eisenberg in [9], Ambainis's algorithm actually gives an $O(N^{k/k+1})$ algorithm for the general k-Subset Finding problem, defined as follows.

k-Subset Finding: Given N elements $x_1, ...x_N \in [M]$, and a k-ary relation $R \subseteq [M]^k$, decide whether there is a k-size set $\{i_1, ..., i_k\}$ *s.t.* $(x_{i_1}, ..., x_{i_k}) \in R$. If yes, output a solution; otherwise reject.

This generalizes Grover's search [11], which can be viewed as the special case of $k = 1$. We can also define the Unique k-Subset Finding problem, which is the same as k-Subset Finding except that it is promised that there is at most one solution set $\{i_1, ..., i_k\}$. As pointed out in [13], by a standard random reduction, we can solve k-Subset Finding with the same complexity as the Unique k-Subset Finding. Therefore, in what follows we mostly study the unique version of the problems.

A lot of recent research in quantum computing is focused on the query models, where the input is accessed by querying an oracle, and the goal is to minimize

* This research was supported in part by NSF grant CCR-0310466.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 430–439, 2005.

the number of the queries made to compute the function. There are mainly two variants of query models. A commonly used one, sometimes called function-evaluation query model, is as follows. A query for the input $x = x_1...x_N$ is represented as

$$|i, b, z\rangle \rightarrow |i, (b + x_i) \ mod \ M, z\rangle \tag{1}$$

where i is the index of the variable that we are currently interested in, b is the value (before the query) in the place where the answer is held, and z is a work state not involved in the current query processing. The other query model is the comparison model, where a query is

$$|i, j, b, z\rangle \rightarrow |i, j, b \oplus \lambda_{x_i \leq x_j}, z\rangle \tag{2}$$

with $b \in \{0, 1\}$ and λ_ϕ being the truth-value of the formula ϕ (throughout the paper). A quantum query computation is a series of operations $U_1, O, U_2, O, ..., U_T$, where each U_i is a unitary operator independent of the input x and O is a query as specified above. We use $Q_2^F(f)$ and $Q_2^C(f)$ to denote the double side bounded error quantum query complexity of f in the function-evaluation model and the comparison model, respectively. For further details on quantum query model, we refer readers to [4, 8] as two surveys.

Ambainis [5] showed that $Q_2^F(f) = O(N^{k/k+1})$ and $Q_2^C(f) = O(N^{k/k+1} \log N)$ for k-Subset Finding. In this paper, we consider two related problems. The first one is to consider what if we know some information about the solution in advance. For example, when $k = 2$, suppose that the unique solution is (i_1, i_2) and we know in advance that i_1 is in some subset of $[N]$. Does this information help our search? To be more precise, consider the following problems.

Unique (m, n) 2-Subset Finding: We are given $x_1, ..., x_N \in [M]$, two sets of indices $J_1, J_2 \subseteq [N]$ with $|J_1| = m, |J_2| = n$, and a relation $R \subseteq [M] \times [M]$, with the promise that there exists at most one pair of $(x_{j_1}, x_{j_2}) \in R$ *s.t.* $j_1 \in J_1$, $j_2 \in J_2$ and $j_1 \neq j_2$. Output the unique pair if it exists, and reject otherwise.

Claw Finding: The above problem with the restrictions that R is the Equality relation and $J_1 \cap J_2 = \emptyset$.

The best previous result for the Claw Finding is given by Buhrman, Durr, Heiligman, Hoyer, Magniez, Santha and de Wolf [7]:

$$\Omega(m^{1/2}) \leq Q_2^C(\text{Claw-Finding}) \leq O((n^{1/2}m^{1/4} + m^{1/2}) \log n) \tag{3}$$

where without loss of generality, they assume $m \geq n$. In this paper, we improve it to the following (almost) tight bounds.

Theorem 1. *For both* Unique (m, n) 2-Subset Finding *and* Claw Finding*, we have*

$$Q_2^F(f) = \Theta((mn)^{1/3} + \sqrt{n} + \sqrt{m}) \tag{4}$$

$$\Omega((mn)^{1/3} + \sqrt{n} + \sqrt{m}) \leq Q_2^C(f) \leq O(((mn)^{1/3} + \sqrt{n} + \sqrt{m})(\log m + \log n)) \tag{5}$$

The proof for the upper bound uses a generalization of Ambainis' quantum walk algorithm [5]. The main difference is that we maintain two sets of registers instead of just one set. The lower bound is shown by a reduction to the $\Omega((n/r)^{1/3})$ lower bound for r-Collision by Shi [2].

We also consider the promised version of the k-Subset Finding problem for general k, and a similar upper bound is given.

The second problem we study is another natural scenario for k-Subset Finding: distributed search. Suppose that Alice has input $x_1, ..., x_n$ and Bob has $y_1, ..., y_n$. Alice can access her input $x_1, ..., x_n$ only by quantum queries as in (1), and she cannot access Bob's input $y_1, ..., y_n$. Symmetric rules apply to Bob. They want to search for the unique pair of (x_i, y_j) in some given relation R, by come communications[1]. In other words, the model is the same as the one used to study quantum communication complexity (see [10]) except that the two parties access their respective inputs by quantum queries (1). So there are two natural resources to consider. One is the number of queries, and the other is the number of qubits in the communication. The former is about quantum query complexity as studied above, and the latter is about quantum communication complexity, introduced by Yao[16] and extensively studied since then (see [10] for a survey). As far as we know, all previous work considers one of these two problems[2]. For example, Ambainis [5] and the first part of this paper consider the quantum query complexity; Buhrman, Cleve and Wigderson [6] show an $O(\sqrt{n}\log n)$ upper bound of quantum communication complexity of Disj, later improved by Hoyer and de Wolf to $O(\sqrt{n}c^{\log^* n})$ [12] and finally to $O(\sqrt{n})$ by Aaronson and Ambainis [1], matching the $\Omega(\sqrt{n})$ lower bound shown by Razborov [15]. Since query and communication are both well-studied resources, it is natural to study both of them simultaneously, and see how they interact with each other.

We can use a protocol similar to the one shown by Buhrman, Cleve and Wigderson [6], but it makes $\Theta(n)$ queries, which is higher than the optimal $\Theta(n^{2/3})$ value. We can also have a protocol achieving the optimal quantum query complexity, but the number of communication qubits is asymptotically more than the optimal $\widetilde{\Theta}(\sqrt{n})$ value. So it seems to exist a tradeoff between the quantum query computation and the quantum communication. This paper gives one tradeoff result as follows. For a protocol P computing function f, denote by $q(P)$ the number of quantum queries and by $c(P)$ the number of communication qubits.

Theorem 2. *Let $f =$ Unique 2-Subset Finding. For any given $q_0 \in (n^{2/3}, n)$, there exists a protocol P with $q(P) = q_0$ and $c(P) = O(\frac{n^2 \log n}{q_0^{3/2}})$.*

[1] If the R is the Equality relation, then the problem is related to Disj, a well-studied function. But we should note that Disj is to decide whether two subsets of an n-element set intersect, while here the distributed search problem is to decide whether two n-element sets intersect.

[2] Some papers study yet other resources. For example, paper [14] gives a lower bound of the tradeoff between communication complexity and round complexity.

In other words, we have a family of protocols with $q(P)^{3/2} \cdot c(P) = O(n^2 \log n)$. This implies that we can pay more for quantum queries to save on quantum communication, and vice versa.

2 The Quantum Query Complexity of Promised Subset Finding

2.1 Review of Ambainis' Search and the Generic Algorithm

We first review Ambainis' search algorithm for UNIQUE k-ELEMENT DISTINCTNESS [5]. The working state is a superposition of basis in the form of $|S, x_S, i\rangle$. Here S is a r-size subset of $[N]$, x_S contains the variable values x_j's for all $j \in S$, and i is an index not in S. An basic tool used in the algorithm is a subroutine called **Quantum Walk** as follows.

Algorithm 1: Quantum Walk on S in A
Input: State $|S, x_S, i\rangle$ and A with $S \subseteq A$, and $i \in A - S$. Suppose that $|S| = r, |A| = N$.

1. $|S, x_S, i\rangle \rightarrow |S, x_S\rangle \left((-1 + \frac{2}{N-r})|i\rangle + \frac{2}{N-r} \sum_{j \in A-S-\{i\}} |j\rangle\right)$
2. $|S, x_S, i\rangle \rightarrow |S \cup \{i\}, x_{S\cup\{i\}}, i\rangle$ by one query.
3. $|S, x_S, i\rangle \rightarrow |S, x_S\rangle \left((-1 + \frac{2}{r+1})|i\rangle + \frac{2}{r+1} \sum_{j \in S-\{i\}} |j\rangle\right)$
4. $|S, x_S, i\rangle \rightarrow |S - \{i\}, x_{S-\{i\}}, i\rangle$ by one query.

An key fact shown by Ambainis [5] is the following. Let $I = \{i_1, ..., i_k\}$ where $(i_1, ..., i_k)$ is the unique k-tuple of equal elements. Define a $(2k+1)$-dimentional subspace

$$\tilde{H} = span\{|\psi_{j,l}\rangle : \ j = 0, ..., k; \ l = 0, 1; \ (j, l) \neq (k, 1)\} \tag{6}$$

where $|\psi_{j,l}\rangle$ is the uniform superposition of states $\{|S, x_S, i\rangle : |S| = r, \ i \in A - S, \ j = |S \cap I|, \ l = \lambda_{i \in I}\}$ (with $\lambda_\phi = 1$ if ϕ is true, and 0 otherwise). Then first, one step of **Quantum Walk** maps $\tilde{H}$ to $\tilde{H}$ itself. Second, the operation of **Quantum Walk**, when restricted on $\tilde{H}$, has $2k + 1$ eigenvalues, one of which is 1 and the corresponding eigenvalue is the starting state $|\psi_{start}\rangle$. The other $2k$ eigenvalues are in the form of $e^{\pm\theta_1 i}, ..., e^{\pm\theta_k i}$, where $\theta_j = (2\sqrt{j} + o(1))/\sqrt{r}$. Though the original k is supposed to be at least 2, we observe that the above fact also holds for case $k = 1$. This will be used in our proof of Theorem 1. Using the following key lemma, Ambainis gave **Algorithm 2** for UNIQUE k-ELEMENT DISTINCTNESS.

Lemma 1 (Ambainis [5]). *Let $\mathcal{H}$ be a finite dimensional Hilbert space and $|\psi_1\rangle$, ..., $|\psi_m\rangle$ be an orthonormal basis for $\mathcal{H}$. Let $|\psi_{good}\rangle$, $|\psi_{start}\rangle$ be two states in $\mathcal{H}$ which are superpositions of $|\psi_1\rangle$, ..., $|\psi_m\rangle$ with real amplitudes and $\langle\psi_{good}|\psi_{start}\rangle = \alpha$. Let U_1, U_2 be unitary transformations on $\mathcal{H}$ satisfying:*

1. *U_1 is the transformation that flips the phase on $|\psi_{good}\rangle$ (*i.e. *$U_1|\psi_{good}\rangle = -|\psi_{good}\rangle$) and leaves any state orthogonal to $|\psi_{good}\rangle$ unchanged.*
2. *U_2 is a transformation which is described by a real-valued $m * m$ matrix in the basis $|\psi_1\rangle$, ..., $|\psi_m\rangle$. Moreover, $U_2|\psi_{start}\rangle = |\psi_{start}\rangle$ and, if $|\psi\rangle$ is an eigenvector of U_2 perpendicular to $|\psi_{start}\rangle$, then $U_2|\psi\rangle = e^{i\theta}|\psi\rangle$ for $\theta \in [\epsilon, 2\pi - \epsilon]$.*

Then, there exists $t = O(\frac{1}{\alpha})$ such that $\langle\psi_{good}|(U_2U_1)^t|\psi_{start}\rangle = \Omega(1)$.

Algorithm 2: for Unique k-Element Distinctness
Input: $x_1, ..., x_N \in [M]$, with the promise that there exists at most one k-size set $I = \{i_1, ..., i_k\} \subseteq [N]$ *s.t.* $x_{i_1} = ... = x_{i_k}$.
Output: I and $x_I = \{x_{i_1}, ..., x_{i_k}\}$ if they exist; otherwise reject.

1. Set up the initial state $|\psi_{start}\rangle = \frac{1}{\sqrt{\binom{N}{r}(N-r)}} \sum_{S\subseteq[N],|S|=r,i\in[N]-S} |S, x_S, i\rangle$.
2. Do $\Theta((\frac{N}{r})^{k/2})$ times
   (a) Check whether $I \subseteq S$. If yes, do the phase flip: $|S, x_S, i\rangle \to -|S, x_S, i\rangle$.
   (b) Do **Quantum Walk** on S in $[N]$ for $\Theta(\sqrt{r})$ times.
3. Measure the resulting state and give the corresponding answer.

By **Lemma 1**, if the (unique) k-size subset I exists, then after Step 2, the state is close to $|\psi_{good}\rangle = \frac{1}{\sqrt{\binom{N-k}{r-k}(N-r)}} \sum_{|S|=r,\ I\subseteq S,\ i\in[N]-S} |S, x_S, i\rangle$, thus the algorithm can output $I = \{i_1, ..., i_k\}$ and $x_I = \{x_{i_1}, ..., x_{i_k}\}$ in Step 3 (with high probability). If such I does not exist, the state after Step 2 is still $|\psi_{start}\rangle$, and thus the algorithm rejects in Step 3.

By letting $r = N^{k/k+1}$, we have an algorithm using $O(N^{k/k+1})$ queries in the function-evaluation model. In comparison model, the upper bound can be achieved with a log factor added [5]. Basically, we keep the set $|S\rangle$ sorted during the computation. So both in the set up phase (Step 1) and in the update phase (Step 2(b)), adding a log factor is enough.

2.2 Proof of Theorem 1

We prove Theorem 1 in this section. For the upper bounds, we give **Algorithm 3**, which refines Ambainis' **Algorithm 2** by maintaining two sets of registers instead of one set.

The following theorem actually shows the upper bound of Theorem 1 in the function-evaluation model.

Theorem 3. *Algorithm 3 outputs the desired results correctly in the function-evaluation model, and we can pick r_1, r_2 to make number of queries be*

$$\begin{cases} O((mn)^{1/3}) & if\ \sqrt{n} \le m \le n^2 \quad (by\ letting\ r_1 = r_2 = (mn)^{1/3}) \\ O(\sqrt{n}) & if\ m < \sqrt{n} \quad (by\ letting\ r_1 = m, r_2 \in [m, (mn)^{1/3}]) \\ O(\sqrt{m}) & if\ m > n^2 \quad (by\ letting\ r_1 \in [n, (mn)^{1/3}], r_2 = n) \end{cases}$$

Algorithm 3: for Unique (m,n) 2-Subset Finding
Input: $x_1, ..., x_N \in [M]$. $J_1, J_2 \subseteq [N]$, $|J_1| = m, |J_2| = n$. $R \subseteq [M] \times [M]$ *s.t.* there is at most one $(x_{j_1}, x_{j_2}) \in R$ with $j_1 \in J_1$, $j_2 \in J_2$ and $j_1 \neq j_2$.
Output: The unique pair (j_1, j_2) if it exists; otherwise reject.

1. Set up the initial state
   $|\psi_{start}\rangle = \frac{1}{\sqrt{T}} \sum_{S_b \subseteq J_b, |S_b| = r_b, i_b \in J_b - S_b} |S_1, x_{S_1}, i_1, S_2, x_{S_2}, i_2\rangle$,
   where $T = \binom{m}{r_1}\binom{n}{r_2}(m - r_1)(n - r_2)$ and $b = 1, 2$.
2. Do $\Theta(\sqrt{\frac{mn}{r_1 r_2}})$ times
   (a) Check whether the unique (j_1, j_2) is in $S_1 \times S_2$. If yes, do the following phase flip: $|S_1, x_{S_1}, i_1, S_2, x_{S_2}, i_2\rangle \to -|S_1, x_{S_1}, i_1, S_2, x_{S_2}, i_2\rangle$.
   (b) Do **Quantum Walk** on S_1 in J_1 for $t_1 = \lceil \frac{\pi}{4}\sqrt{r_1} \rceil$ times.
   Do **Quantum Walk** on S_2 in J_2 for $t_2 = \lceil \frac{\pi}{8}\sqrt{r_2} \rceil$ times.
3. Measure the resulting state and give the corresponding answer.

Proof. Correctness: First, if there is no desired pair, then the algorithm actually does nothing, so the state after Step 2 is still $|\psi_{start}\rangle$. Thus in Step 3, we cannot find the desired pair after the measurement, and we will reject.

On the other side, if there is the pair, we shall use **Lemma 1** to show that we can find it. Suppose $(j_1, j_2) \in J_1 \times J_2$ is the desired pair. First, define $\tilde{H}_1$ as in (6), with $|\psi_{j,l}\rangle$ being the uniform superposition of states $\{|S_1, x_{S_1}, i_1\rangle : S_1 \subseteq J_1, |S_1| = r_1,\ i_1 \in J_1 - S_1,\ j = \lambda_{j_1 \in S_1},\ l = \lambda_{i_1 = j_1}\}$. Note that it is exactly the "$k = 1$" case of (6), so W_1, the operator of **Quantum Walk** on S_1 in J_1, when restricted on $\tilde{H}_1$, has 3 eigenvalues. One of the eigenvalues is 1, and the corresponding eigenvector is $|\psi_{start,1}\rangle = \frac{1}{\sqrt{\binom{m}{r_1}(m-r_1)}} \sum_{S_1 \subseteq J_1, |S_1| = r_1, i_1 \in J_1 - S_1} |S_1, x_{S_1}, i_1\rangle$. The other two eigenvalues are $e^{\pm i\theta_1}$, and $\theta_1 = (2 + o(1))/\sqrt{r_1}$. Therefore, $W_1^{t_1}$ has 3 eigenvalues: 1 (with the eigenvector $|\psi_{start,1}\rangle$) and $e^{\pm i\theta_1'}$ where $\theta_1' = \frac{\pi}{2} + o(1)$.

$\tilde{H}_2$ is defined symmetrically, as well as W_2, $|\psi_{start,2}\rangle$ and θ_2. As a result, $W_2^{t_2}$ has 3 eigenvalues: 1 (with the eigenvector $|\psi_{start,2}\rangle$) and $e^{\pm i\theta_2'}$ where $\theta_2' = \frac{\pi}{4} + o(1)$. The whole step 2(b) restricted on $\tilde{H}_1 \otimes \tilde{H}_2$ is the operation $W = (I_1 \otimes W_2)(W_1 \otimes I_2)$. Now note that the eigenvalues of W are given by

$$\{\lambda \cdot \mu : \lambda \text{ is an eigenvalue of } W_1 \text{ on } \tilde{H}_1, \text{ and } \mu \text{ is an eigenvalue of } W_2 \text{ on } \tilde{H}_2\}.$$

Therefore, W has 9 eigenvalues $\{e^{i(b_1\theta_1' + b_2\theta_2')} : b_1, b_2 \in \{-1, 0, 1\}\}$. It is easy to check that one of eigenvalues is 1, and the corresponding eigenvector is $|\psi_{start,1}\rangle \otimes |\psi_{start,2}\rangle$, which is exactly the $|\psi_{start}\rangle$ in Algorithm 3. All the other 8 eigenvalues are in the form of $e^{\pm i\theta}$, for some $\theta \in [\pi/4 - o(1), 2\pi - \pi/4 + o(1)]$. Finally, we calculate $\alpha = \langle\psi_{start}|\psi_{good}\rangle$: $\alpha = \sqrt{\mathbf{Pr}_{|S_1|=r_1, |S_2|=r_2}[(j_1, j_2) \in S_1 \times S_2]} = \Theta(\sqrt{\frac{r_1 r_2}{mn}})$. So the number of iterations in Step 2 is $1/\alpha = \Theta(\sqrt{\frac{mn}{r_1 r_2}})$ and the correctness holds by **Lemma 1**.

It is easy to verify that the number of queries used is $O(r_1 + r_2 + \sqrt{\frac{mn}{r_1 r_2}}(\sqrt{r_1} + \sqrt{r_2})) = O(r_1 + r_2 + \frac{\sqrt{mn}}{\sqrt{r_1}} + \frac{\sqrt{mn}}{\sqrt{r_2}})$. Now we need minimize it, with restrictions

$r_1 \leq m$ (because S_1 is a subset of J_1) and $r_2 \leq n$. For the $(r_1 + \frac{\sqrt{mn}}{\sqrt{r_1}})$ part, it is not hard to see that if $m \geq \sqrt{n}$ then $\min_{r_1 \leq m}(r_1 + \frac{\sqrt{mn}}{\sqrt{r_1}}) = (mn)^{1/3}$ and the minimum is achieved when $r_1 = (mn)^{1/3}$; otherwise $\min_{r_1 \leq m}(r_1 + \frac{\sqrt{mn}}{\sqrt{r_1}}) = m + \sqrt{n}$ and the minimum is obtained when $r_1 = m$. Analyze the $r_2 + \sqrt{mn}/\sqrt{r_2}$ part similarly, and we can get the conclusion as in the statement of the theorem. □

Next we prove the lower bound part in Theorem 1. Note that since Claw-Finding is a special case of (m, n) 2-Subset Finding, it is enough to show the lower bound for Q_2(Claw-Finding).

Proof. It is sufficient to prove the lower bound of $\Omega((mn)^{1/3})$. We will show it by a reduction to the 2-Collision problem, which is to distinguish whether a function $f : [N] \to [N]$ is one-to-one or two-to-one. This problem is shown by Aaronson, Shi [2] and Ambainis [3] to have $\Omega(N^{1/3})$ lower bound of quantum query complexity. Assume that we can solve (m, n) 2-Subset Finding with $o((mn)^{1/3})$ queries, then we can have an $o(N^{1/3})$ algorithm for the 2-Collision problem as follows. Let $f : [N] \to [N]$ be a function, where $N = mn$, and we are to decide whether it is one-to-one or two-to-one. First pick a random set $S_1 \subseteq [N]$ of size m and then pick another random set $S_2 \subseteq [N] - S_1$ of size n. If f is one-to-one, then $f(i_1) \neq f(i_2)$ for any $i_1 \in S_1$ and $i_2 \in S_2$, since $S_1 \cap S_2 = \emptyset$. On the other hand, if f is two-to-one, then by a standard probability calculation we know that with constant probability there will be $i_1 \in S_1$ and $i_2 \in S_2$ such that $f(i_1) = f(i_2)$. Therefore, whether f is two-to-one or one-to-one is, up to a constant probability, equivalent to whether there are $i_1 \in S_1$ and $i_2 \in S_2$ such that $f(i_1) = f(i_2)$, which can be decided with $o((mn)^{1/3}) = o(N^{1/3})$ queries, by our assumption. This contradicts to the $\Omega(N^{1/3})$ lower bound of 2-Collision [2, 3], so Q_2^F(Unique (m, n) 2-Subset Finding) $= \Omega((mn)^{1/3})$. □

We make a few remarks about Claw-Finding problem in the comparison model in **Theorem 1** to end the subsection. The upper bound of Q_2^C(Claw-Finding) is got in the same way we described at the end of Section 2.1, with only a $\log n$ factor added. As to the lower bound, since we can use 2 queries in the function-evaluation model to simulate 1 query in comparison model, we have always $Q_2^F(f) \leq 2Q_2^C(f)$. So a lower bound for $Q_2^F(f)$ is also a lower bound for $Q_2^C(f)$ up to a factor of 2.

2.3 The General Case

We can use the same technique to give a generic algorithm for a general promised subset finding problem.

Unique $(n_i, k_i)_{i=1,...,l}$ k-Subset Finding: We are given $x_1, ..., x_N \in [M]$, l sets of indices $J_1, ..., J_l \subseteq [N]$ with $|J_i| = n_i$ $(i = 1, ..., l)$, and a relation $R \subseteq [M]^k$, where $k = \sum_{i=1}^{l} k_i$ is constant, with the promise that there is at most one k-size set $\{j_{11}, ..., j_{1k_1}, ..., j_{l1}, ..., j_{lk_l}\}$ *s.t.* $(x_{j_{11}}, ..., x_{j_{1k_1}}, ..., x_{j_{l1}}, ..., x_{j_{lk_l}}) \in R$

and $j_{ip} \in J_i$ $(i = 1, ..., l; p = 1, ..., k_i)$. Output the unique k-set if it exists; otherwise reject.

If R is Equality relation, we call the problem UNIQUE $(n_i, k_i)_{i=1,...,l}$ k-ELEMENT DISTINCTNESS. An generic algorithm for UNIQUE $(n_i, k_i)_{i=1,...,l}$ k-SUBSET FINDING is as follows. As in [13], we use three kinds of registers: set registers S, data registers $D(s)$ and coin register c.

Algorithm 4: for Unique $(n_i, k_i)_{i=1,...,l}$ k-Subset Finding
Input: $x_1, ..., x_N \in [M]$, $J_1, ..., J_l \subseteq [N]$ with $|J_i| = n_i$ $(i = 1, ..., l)$, $R \subseteq [M]^k$, where $k = \sum_{i=1}^{l} k_i$ is constant, with the promise that there is at most one k-size set $J = \{j_{11}, ..., j_{1k_1}, ..., j_{l1}, ..., j_{lk_l}\}$ *s.t.* $(x_{j_{11}}, ..., x_{j_{1k_1}}, ..., x_{j_{l1}}, ..., x_{j_{lk_l}}) \in R$ and $j_{ip} \in J_i$ $(i = 1, ..., l; p = 1, ..., k_i)$.
Output: Output the unique k-set J if it exists; reject otherwise.

1. Create the state $\sum_{S_i \subseteq J_i, |S_i| = r_i, c_i \in J_i - S_i} |S_1, c_1\rangle ... |S_l, c_l\rangle$.
2. Get the data $D(S_i)$ for each S_i. Then the state is
$$\sum_{S_i \subseteq J_i, |S_i| = r_i, c_i \in J_i - S_i} |S_1, D(S_1), c_1\rangle ... |S_l, D(S_l), c_l\rangle.$$
3. Do $\Theta(\frac{\prod_{i=1}^{l} n_i^{k_i/2}}{\prod_{i=1}^{l} r_i^{k_i/2}})$ times
   (a) If $\boldsymbol{j} \in S_1^{k_1} \times ... \times S_l^{k_l}$, then do phase flip; else do nothing.
   (b) For $i = 1, ..., l$: do **Quantum Walk** on S_i in J_i for $\lceil \sqrt{r_i}\pi/2^{i+1} \rceil$ times .
4. Measure the resulting state and give the corresponding answer.

Suppose the setup step (Step 2) takes $s(r_1, ..., r_l)$ queries, check step (Step 3(a)) takes $c(r_1, ..., r_l)$ queries, and each **Quantum Walk** on $|S_i, D(S_i), c_i\rangle$ in J_i takes $u(r_i)$ to update the data $D(S_i)$. Using the similar analysis as in the proof for Theorem 1, we can show the following upper bound for UNIQUE $(n_i, k_i)_{i=1,...,l}$ k-SUBSET FINDING and UNIQUE $(n_i, k_i)_{i=1,...,l}$ k-ELEMENT DISTINCTNESS. The only thing needed to note is that the operator of Step 3(b) has eigenvalue $\{e^{i\theta} : \theta = b_1\frac{\pi}{2} + b_2\frac{\pi}{4} + ... + b_l\frac{\pi}{2^{l+1}} + o(1), b_1, ..., b_l \in \{-1, 0, 1\}\}$. But it is easy to check that for any $b_1, ..., b_l \in \{-1, 0, 1\}$ such that b_i's are not all zeros, it holds that $\frac{\pi}{2^{l+1}} \leq |b_1\frac{\pi}{2} + b_2\frac{\pi}{4} + ... + b_l\frac{\pi}{2^{l+1}}| < \pi$, so we can use the Lemma 1 and the proof passes through.

Theorem 4. *Algorithm 4 has quantum query complexity*
$$O(s(r_1, ..., r_l) + \frac{\prod_{i=1}^{l} n_i^{k_i/2}}{\prod_{i=1}^{l} r_i^{k_i/2}}(c(r_1, ..., r_l) + \sqrt{r_1}u(r_1) + ... + \sqrt{r_l}u(r_l)).$$
In particular, if $s(r_1, ..., r_l) = \sum_i r_i$, $c(r_1, ..., r_l) = 0$ *and* $u(r_i) = 1$ *as in* UNIQUE $(n_i, k_i)_{i=1,...,l}$ k-ELEMENT DISTINCTNESS *problem, then the complexity is*
$$O(\sum_i r_i + \frac{\prod_{i=1}^{l} n_i^{k_i/2}}{\prod_{i=1}^{l} r_i^{k_i/2}}(\sum_i \sqrt{r_i})).$$

When $(\prod_{i=1}^{l} n_i^{k_i})^{\frac{1}{k+1}} \leq n_i$ *is satisfied* $(i = 1, ..., l)$*, we can pick* $r_i = (\prod_{i=1}^{l} n_i^{k_i})^{\frac{1}{k+1}}$*, and the query complexity is* $O((\prod_{i=1}^{l} n_i^{k_i})^{\frac{1}{k+1}})$.

3 Tradeoff Between Quantum Query and Communication

In this section we prove **Theorem 3** by giving a family of protocols achieving the tradeoff result. Note that in **Algorithm 3**, both the preparation of the initial state $|\psi_{start}\rangle$ in Step 1 and the Quantum Walks in Step 2(b) can be done distributively. So it naturally induces a communication protocol as follows.

Protocol 1: for distributed Unique 2-Subset Finding
Input: $x_1, ..., x_N \in [M]$. $J_1, J_2 \subseteq [N]$, $|J_1| = m, |J_2| = n$. $R \subseteq [M] \times [M]$ *s.t.* there is at most one $(x_{j_1}, x_{j_2}) \in R$ with $j_1 \in J_1$, $j_2 \in J_2$ and $j_1 \neq j_2$.
Output: The unique pair (j_1, j_2) if it exists; otherwise reject.

1. Alice sets up her initial state
   $|\psi_a\rangle = \frac{1}{\sqrt{\binom{n}{r_1}(n-r_1)}} \sum_{S_1 \subseteq J_1, |S_1|=r_1, i_1 \in J_1 - S_1} |S_1, x_{S_1}, i_1\rangle$ in her register R_a
   Bob sets up his initial state
   $|\psi_b\rangle = \frac{1}{\sqrt{\binom{n}{r_2}(n-r_2)}} \sum_{S_2 \subseteq J_2, |S_2|=r_2, i_2 \in J_2 - S_2} |S_2, x_{S_2}, i_2\rangle$ in his register R_b
2. Do $\Theta(\frac{n}{\sqrt{r_1 r_2}})$ times
   (a) Bob sends R_b (*i.e.* all his qubits) to Alice.
   (b) Alice checks whether $(j_1, j_2) \in S_1 \times S_2$. If yes, do the following phase flip: $|S_1, x_{S_1}, i_1, S_2, x_{S_2}, i_2\rangle \to -|S_1, x_{S_1}, i_1, S_2, x_{S_2}, i_2\rangle$.
   (c) Alice sends R_b back to Bob.
   (d) Alice does $\lceil \frac{\pi}{4}\sqrt{r_1} \rceil$ times **Quantum Walk** on S_1 in J_1.
   Bob does $\lceil \frac{\pi}{8}\sqrt{r_2} \rceil$ times **Quantum Walk** on S_2 in J_2.
3. Bob does the measurement and outputs the corresponding result.

The correctness of the protocol is obvious because it is essentially the same as Algorithm 3. We now analyze the complexity. The number of queries is the same as that of Algorithm 3, *i.e.* $q(P) = \Theta(r_1 + r_2 + \frac{n}{\sqrt{r_1 r_2}}(\sqrt{r_1} + \sqrt{r_2})) = \Theta(r_1 + r_2 + n(1/\sqrt{r_1} + 1/\sqrt{r_2}))$. The number of communication qubits of this protocol is $c(P) = \Theta(\frac{n}{\sqrt{r_1 r_2}} r_2 \log n) = \Theta(\sqrt{\frac{r_2}{r_1}} n \log n)$. If $t = r_1/r_2 \geq 1$, then $q(P) = \Theta(r_1 + n/\sqrt{r_2}) = \Theta(t r_2 + n/\sqrt{r_2}) \geq \Theta(t^{1/3} n^{2/3})$, and the equality is achieved when $r_2 = (n/t)^{2/3}$. So for any given $q_0 \in (n^{2/3}, n)$, let $r_1 = q_0$ and $r_2 = n^2/q_0^2$, then $q(P) = \Theta(q_0)$ and $c(P) = \Theta(\frac{n^2 \log n}{q_0^{3/2}})$.

4 Conclusion

We show a generalization of the recent quantum search algorithms [5, 9, 13] by using more sets of registers. We hope that it can serve as a building block

for other problems. It will be especially interesting if the algorithm can attack problems which are not given as a promised ones. For example, can the ideas of this paper be used to improve the $O(n^{1.3})$ upper bound [13] for Triangle?

References

1. S. Aaronson and A. Ambainis. Quantum search of spatial regions. Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 200-209, 2003. Earlier version at quant-ph/0303041
2. S. Aaronson, Y. Shi. Quantum lower bounds for the collision and the element distinctness problems. Journal of the ACM, 51(4), pp. 595-605, 2004. Earlier version at STOC 2002 and FOCS 2002, also at quant-ph/0111102 and quant-ph/0112086.
3. A. Ambainis. Quantum lower bounds for collision and element distinctness with small range. Theory of Computing, 1(3), 2005. Earlier version at quant-ph/0305179
4. A. Ambainis. Quantum query algorithms and lower bounds, Proceedings of FOTFS III, to appear
5. A. Ambainis. Quantum walk algorithm for element distinctness. Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 22-31, 2004. Earlier version at quant-ph/0311001
6. H. Buhrman, R. Cleve, A. Wigderson. Quantum vs. classical communication and computation. Proceedings of the 30th Annual ACM Symposium on Theory of Computing, pp. 63-68, 1998
7. H. Buhrman, C. Durr, M. Heiligman, P. Hoyer, F. Magniez, M. Santha, R. de Wolf. Quantum algorithms for Element Distinctness. Proceedings of Sixteenth IEEE conference on Computational Complexity, pp. 131-137, 2001. Journal version to appear in SIAM Journal of Computing.
8. H. Buhrman, R. de Wolf. Complexity measures and decision tree complexity: a survey. Theoretical Computer Science, 288(1), pp. 21-43, 2002
9. A. Childs and J. Eisenberg. Quantum algorithms for subset finding. quant-ph/0311038
10. R. de Wolf. Quantum communication and complexity. Theoretical Computer Science, 287(1), pp. 337-353, 2002.
11. L. Grover. A fast quantum mechanical algorithm for database search. Proceedings of the 28th Annual ACM Symposium on Theory of Computing, pp. 212-219, 1996
12. P. Hoyer and R. de Wolf. Improved quantum communication complexity bounds for disjointness and equality. Proceedings of the 19th Symposium on Theoretical Aspects of Computer Science, pp. 299-310, 2002. Earlier version at quant-ph/0109068.
13. F. Magniez, M. Santha, M. Szegedy. Quantum algorithms for the Triangle problem. Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 1109-1117, 2005. Earlier versions at quant-ph/0310107 and quant-ph/0310134
14. R. Jain. J. Radhakrishnan, P. Sen. A lower bound for bounded round quantum communication complexity of set disjointness. Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, pp. 220 - 229, 2003
15. A. Razborov. Quantum communication complexity of symmetric predicates. Izvestiya: Mathematics, 67(1), pp. 145-159, 2003
16. A. Yao, Quantum circuit complexity, Proceedings of the 34th IEEE Symposium on Foundations of Computer Science, pp. 352-361, 1993

Efficient and Simple Generation of Random Simple Connected Graphs with Prescribed Degree Sequence

Fabien Viger[1,2] and Matthieu Latapy[2]

[1] LIP6, University Pierre and Marie Curie, 4 place Jussieu, 75005 Paris
[2] LIAFA, University Denis Diderot, 2 place Jussieu, 75005 Paris
{fabien,latapy}@liafa.jussieu.fr

Abstract. We address here the problem of generating random graphs uniformly from the set of simple connected graphs having a prescribed degree sequence. Our goal is to provide an algorithm designed for practical use both because of its ability to generate very large graphs (efficiency) and because it is easy to implement (simplicity).
We focus on a family of heuristics for which we prove optimality conditions, and show how this optimality can be reached in practice. We then propose a different approach, specifically designed for typical real-world degree distributions, which outperforms the first one. Assuming a conjecture, we finally obtain an $O(n \log n)$ algorithm, which, in spite of being very simple, improves the best known complexity.

1 Introduction

Recently, it appeared that the degree distribution of most real-world complex networks is well approximated by a power law, and that this unexpected feature has a crucial impact on many phenomena of interest [5]. Since then, many models have been introduced to capture this feature. In particular, the Molloy and Reed model [13], on which we will focus, generates a random graph with prescribed degree sequence in linear time. However, this model produces graphs that are neither *simple*[1] nor *connected*. To bypass this problem, one generally simply removes multiple edges and loops, and then keeps only the largest connected component. Apart from the expected size of this component [2, 14], very little is known about the impact of these removals on the obtained graphs, on their degree distribution and on the simulations processed using them.

The problem we address here is the following: given a degree sequence, we want to generate a random *simple connected* graph having exactly this degree sequence. Moreover, we want to be able to generate very large such graphs, typically with more than one million vertices, as often needed in simulations.

Although it has been widely investigated, it is still an open problem to directly generate such a random graph, or even to enumerate them in polynomial time, even without the connectivity requirement [11, 12].

[1] A simple graph has neither multiple edges, *i.e.* several edges binding the same pair of vertices, nor loops, *i.e.* edges binding a vertex to itself.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 440–449, 2005.

In this paper, we will first present the best solution proposed so far [6, 12], discussing both theoretical and practical considerations. We will then deepen the study of this algorithm, which will lead us to an improvement that makes it optimal among its family. Furthermore, we will propose a new approach solving the problem in $O(n \log n)$ time, and being very simple to implement.

2 Context

The Markov Chain Monte-Carlo Algorithm

Several techniques have been proposed to solve the problem we address. We will focus here on the Markov chain Monte-Carlo algorithm [6], pointed out recently by an extensive study [12] as the most efficient one.

The generation process is composed of three main steps:

1. **Realize the sequence**: generate a simple graph that matches the degree sequence,
2. **Connect** this graph, without changing its degrees, and
3. **Shuffle** the edges to make it random, while keeping it connected and simple.

The Havel-Hakimi algorithm [7, 8] solves the first step in linear time and space. A result of Erdös and Gallai [4] shows that this algorithm succeeds if and only if the degree sequence is realizable.

The second step is achieved by swapping edges to merge separated connected components into a single connected component, following a well-known graph theory algorithm [3, 15]. Its time and space complexities are also linear.

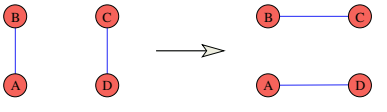


Fig. 1. Edge swap

The third step is achieved by randomly swapping edges of the graph, checking at each step that we keep the graph simple and connected. Given the graph G_t at some step t, we pick two edges at random, and then we swap them as shown in Figure 1, obtaining another graph G' with the same degrees. If G' is still simple and connected, we consider the swap as *valid*: $G_{t+1} = G'$. Otherwise, we reject the swap: $G_{t+1} = G_t$

This algorithm is a Markov chain where the **space** S is the set of all simple connected graphs with the given degree sequence, the **initial state** G_0 is the graph obtained by the first two steps, and the **transition** $G_i \to G_j$ has probability $\frac{1}{m(m-1)}$ if there exists an edge swap that transforms G_i in G_j. If there are no such swap, this transition has probability 0 (note that the probability of

the transition $G_i \to G_i$ is given by the number of invalid swaps on G_i divided by $m(m-1)$).

We will use the following known results:

Theorem 1 *This Markov chain is irreducible [15], symmetric [6], and aperiodic [6].*

Corollary 2 *The Markov chain converges to the uniform distribution on every states of its space,* i.e. *all graphs having the wanted properties.*

These results show that, in order to generate a random graph, it is sufficient to do *enough* transitions. No formal result is known about the convergence speed of the Markov chain, *i.e.* the required number of transitions. However, massive experiments [6, 12] applied the shuffle process with an extremely biased G_0 and showed clearly that $O(m)$ edge swaps are sufficient, by comparing a large set of non-trivial metrics (such as the diameter, the flow, and so on) over the sampled graphs and random graphs. Moreover, we proved[2] that for any *non-ill shaped*[4] degree distribution, the ratio of *valid* edge swaps is greater than some positive constant, so that $O(m)$ transitions are sufficient to ensure $\Omega(m)$ swaps to be done. Therefore, we will assume the following:

Empirical Result 1 *[6, 12] The Markov chain converges after $O(m)$ transitions.*

Complexity

The first two steps of the random generation (realization of the degree sequence and connection of the graph) are done in $O(m)$ time and space. Using hash tables for the adjacency lists, each transition may be done in $O(1)$ time, to which we must add the connectivity tests that take $O(m)$ time per transition. Thus, the total time complexity for the shuffle is quadratic:

$$C_{naive} = O(m^2) \tag{1}$$

Using the structures described in [9, 10, 17] to maintain connectivity in dynamic graphs, one may reduce this complexity to the much smaller:

$$C_{dynamic} = O\left(m \log n (\log \log n)^3\right) \tag{2}$$

Notice however that these structures are quite intricate, and that the constants are large for both time and space complexities. The naive algorithm, despite the fact that it runs in $O(m^2)$ time, is therefore generally used in practice since it has the advantage of being extremely easy to implement. Our contribution in this paper will be to show how it can be significantly improved while keeping it very simple, and that it can even outperform the dynamical algorithm.

[2] All the proofs, and more details may be found in the full version [18].

Speed-Up and the Gkantsidis et al. Heuristics

Gkantsidis et al. proposed a simple way to speed-up the naive implementation [6]: instead of running a connectivity test for *each* transition, they do it every T transitions, for some integer $T \geq 1$ called the *speed-up window*. If the graph obtained after these T transitions is not connected anymore, the T transitions are cancelled.

They proved that this process still converges to the uniform distribution, although it is no longer composed of a single Markov chain but of a concatenation of Markov chains [6]. The global time complexity of connectivity tests C_{conn} is reduced by a factor T, but at the same time the swaps are more likely to get cancelled: with T swaps in a row, the graph has more chances to get disconnected than with a single one. Let us introduce the following quantity:

Definition 1 (Success rate) *The success rate $r(T)$ of the speed-up at a given step is the probability that the graph obtained after T swaps is still connected.*

The shuffle process now requires $O(m/r(T))$ transitions. The time complexity therefore becomes:

$$C_{Gkan} = O\left(r(T)^{-1}\left(m + \frac{m^2}{T}\right)\right) \tag{3}$$

Notice that there is a trade-off between the idea of reducing the connectivity test complexity and the increase of the required number of transitions. To bypass this problem, Gkantsidis et al. used the following heuristics:

Heuristics 1 (Gkantsidis et al. heuristics) `IF` *the graph got disconnected after T swaps* `THEN` $T \leftarrow T/2$ `ELSE` $T \leftarrow T+1$

3 More from the Gkantsidis et al. Heuristics

The problem we address now is to estimate the efficiency of the Gkantsidis heuristics. First, we introduce a framework to evaluate the ideal value for the window T. Then, we analyze the behavior of the Gkantsidis et al. heuristics, and get an estimation of the difference between the speed-up factor they obtain and the optimal speed-up factor. We finally propose an improvement of this heuristics which reaches the optimal. We also provide experimental evidences for the obtained performance.

The Optimal Window Problem

We introduce the following quantity:

Definition 2 (Disconnection probability) *Given some graph G, the* disconnection probability *p is the probability that the graph becomes disconnected after a random edge swap.*

Hypothesis 1 *The disconnection probability p is constant during T consecutive swaps*

Hypothesis 2 *The probability that a disconnected graph gets connected with a random edge swap, called the* reconnection probability*, is equal to zero.*

These hypothesis are reasonable approximations in our context and will actually be confirmed in the following. Thanks to them, we get the following expression for the success rate $r(T)$, which is the probability that the graph stays connected after T swaps:

$$r(T) = (1-p)^T \tag{4}$$

Definition 3 (Speed-up factor) *The* speed-up factor $\theta(T) = T \cdot r(T)$ *is the expectation of the number of swaps actually performed (not counting cancelled swaps) between two connectivity tests.*

The speed-up factor $\theta(T)$ represents the *actual* gain induced by the speed-up for the total complexity of the connectivity tests C_{conn}.

Now, given a graph G with disconnection probability p, the *best* window T is the window that maximizes the speed-up factor $\theta(T)$. We find an optimal value $T = -1/\ln(1-p)$, which corresponds to a success rate $r(T) = 1/e$. Finally, we obtain the following theorem:

Theorem 3 *The speed-up factor θ_{max} is reached if and only if one of the equivalent conditions is satisfied:*

$$\textbf{(i)}\ T = (-\ln(1-p))^{-1} \qquad \textbf{(ii)}\ r(T) = e^{-1}$$

The value of θ_{max} depends only on p and is given by

$$\theta_{max} = (-\ln(1-p)\cdot e)^{-1} \quad \sim_{p\to 0} \quad (p\cdot e)^{-1}$$

Analysis of the Heuristics

Knowing the optimality condition, we tried to estimate the performance of the Gkantsidis et al. heuristics. Considering p as asymptotically small, we obtained[4] the following:

Theorem 4 *The speed-up factor $\theta_{Gkan}(p)$ obtained with the Gkantsidis heuristics verifies:*

$$\forall \epsilon > 0, \qquad \theta_{Gkan} = o\left((\theta_{max})^{1/2+\epsilon}\right) \quad \textit{when} \quad p \to 0$$

More intuitively, this comes from the fact that the Gkantsidis et al. heuristics is too pessimistic: when the graph gets disconnected, the decrease of T is too strong; conversely, when the graph stays connected, T grows too slowly. By doing so, one obtains a very high success rate (very close to 1), which is not the optimal (see Theorem 3).

An Optimal Dynamics

To improve the Gkantsidis et al. heuristics we propose the following one (with two parameters q^- and q^+) :

Heuristics 2 `IF` *the graph got disconnected after* T *swaps* `THEN` $T \leftarrow T \cdot (1 - q^-)$ `ELSE` $T \leftarrow T \cdot (1 + q^+)$

The main idea was to avoid the linear increase in T, which is too slow, and to allow more flexibility between the two factors $1 - q^-$ and $1 + q^+$. We proved[4] the following:

Theorem 5 *With this heuristics, a constant* p*, and for* q^+, q^- *close enough to* 0*, the window* T *converges to the optimal value and stays arbitrarily close to it with arbitrarily high probability if and only if*

$$q^+/q^- = e - 1 \tag{5}$$

Experimental Evaluation of the New Heuristics

To evaluate the relevance of these results, based on Hypothesis 1 and 2, we will now compare empirically the speed-up factors θ_{Gkan}, θ_{new} and θ_{best} respectively obtained with the three following heuristics:

1. The Gkantsidis et al. heuristics (Heuristics 1)
2. Our new heuristics (Heuristics 2)
3. The *optimal* heuristics: at every step, we compute the window T giving the maximal speed-up factor θ_{best} [3].

We generated random graphs with various heavy tailed[4] degree sequences, using a wide set of parameters, and all the results were consistent with our analysis: θ_{Gkan} behaved asymptotycally like $\sqrt{\theta_{best}}$, and our average speed-up factor θ_{new} always reached at least 90% of the optimal θ_{best}. Some typical results are shown below.

These experiments show that our new heuristics is very close to the optimal. Thus, despite the fact that p actually varies during the shuffle, our heuristics react fast enough (in regard to the variations of p) to get a good, if not optimal, window T. We therefore obtain a success rate $r(T)$ in a close range around e^{-1}.

These empirical evidences confirm the validity of our formal approach. We obtained a total complexity $C_{new} = O(m + \mathbf{p} \cdot m^2)$, instead of the already improved $C_{Gkan} = O(m + \sqrt{\mathbf{p}} \cdot m^2)$. Despite the fact that it is asymptotically still outperformed by the complexity of the dynamic connectivity algorithm $C_{dynamic}$ (see Eq. 2), C_{new} may be smaller in practice if p is small enough. For many graph topologies corresponding to real-world networks, especially the dense ones (like social relations, word co-occurences, WWW), and therefore a low disconnection probability, our algorithm represents an alternative that may behave faster, and which implementation is much easier.

[3] The heavy cost of this prohibits its use, as a heuristics. It only serves as a reference.

[4] We used power-law like distributions: $P(X = k) = (k + \mu)^{-\alpha}$, where α represents the "heavy tail" behavior, while μ can be tuned to obtain the desired average z.

Table 1. Average speed-up factors for various values of the average degree z, and for graphs of size $n = 10^4$

$\alpha = 2.5$

z	θ_{Gkan}	θ_{new}	θ_{best}
2.1	0.79	0.88	0.90
3	3.00	5.00	5.19
6	20.9	112	117
12	341	35800	37000

$\alpha = 3$

z	θ_{Gkan}	θ_{new}	θ_{best}
2.1	1.03	1.20	1.26
3	5.94	12.3	12.4
6	32.1	216	234
12	578	89800	91000

4 A Log-Linear Algorithm?

We will now show that in the particular case of heavy-tailed degree distributions like the ones met in practice [5], one may reduce the disconnection probability p at logarithmic cost, thus reducing dramatically the complexity of the connectivity tests.

Guiding Principle

In a graph with a heavy-tailed degree distribution, most vertices have a low degree. This means in particular that, by swapping random edges, one may quite easily create very small isolated component. Conversely, the non-negligible number of vertex of high degree form a robust core, so that it is very unlikely that a random swap creates two large disjoint components.

Definition 4 (Isolation test) *An isolation test of width K on vertex v tests wether this vertex belongs to a connected component of size lower than or equal to K.*

To avoid the disconnection, we will now perform an isolation test after every transition. If this isolation test returns `true`, we cancel the swap rightaway. This way, we detect, at low cost $O(K)$, a significant part of the disconnections.

The disconnection probability p is now the probability that after T swaps *which passed the isolation test*, the graph gets disconnected. It is straightforward to see that p is decreasing with K; more precisely, strong empirical evidences and formal arguments[4] led us to the following conjecture:

Conjecture 1 *The disconnection probability p for random connected graphs with heavy-tailed degree distributions decreases* **exponentially** *with K: $p(K) = O(e^{-\lambda K})$ for some positive constant λ depending on the distribution, and not on the size of the graph.*

The Final Algorithm

Let us introduce the following quantity:

Definition 5 (Characteristic isolation width) *The characteristic isolation width K_G of a graph G having m edges is the minimal* isolation test width K *such that the disconnection probability $p(K)$ verifies $p(K) < 1/m$.*

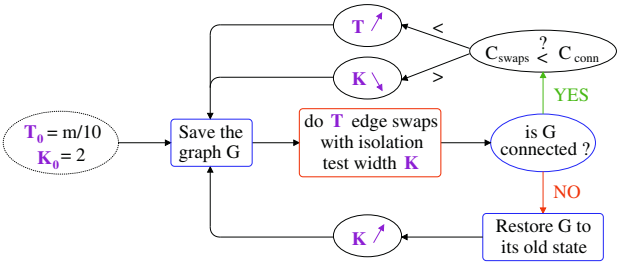


Fig. 2. Our final heuristics used to adjust K and T

Now, we can apply the shuffle process, as seen before, but with a window $T = \Theta(m)$, and an isolation test width K equal to K_G. From Conjecture 1 and the definition of K_G, we deduce that this process will perform $\Omega(m)$ swaps in $O(m \log n)$ time. The difficulty might be to guess K_G. We showed[4] that the heuristics shown in Figure 2 solves this: it aims at equilibrating C_{swaps} and C_{conn} by dynamically adjusting K and T, looking for a high success rate $r(K, T)$ and keeping a large window $T = \Omega(m)$.

We compare in Table 2 typical running times with the naive algorithm, the Gkantsidis et al. heuristics, our improved version of this heuristics, and our final algorithm. Implementations are provided at [18].

Table 2. Average time for the generation of graphs of various sizes with the same heavy-tailed degree distribution ($\alpha = 2.5$, $z = 6.7$) on a Centrino 1.5GHz with 512MB RAM

m	Naive	Gkan. heur.	Heuristics 2	Final algo.
10^3	0.51s	0.02s	0.02s	0.02s
10^4	26.9s	1.15s	0.47s	0.08s
10^5	3200s	142s	48s	1.1s
10^6	$\approx 4 \cdot 10^5$s	$\approx 3 \cdot 10^4$s	10600s	25.9s
10^7	$\approx 4 \cdot 10^7$s	$\approx 3 \cdot 10^6$s	$\approx 10^6$s	420s

Towards a $O(m \log \log n)$ Algorithm?

The isolation tests are typically breadth- or depth-first searches that stop when they have visited $K+1$ vertices, or when then have explored a component of size S lower than K. In the latter case, Conjecture 1 ensures[4] that the expectation of S is $< S >= O(1)$, so that the average complexity of the isolation test was also $O(1)$. Taking advantage of the heavy-tailed degree distribution, we may be

able to reduce as well the complexity of the isolation tests that do not detect a disconnection.

The idea is simple: if the search meets a vertex of degree greater than K, it can stop, because it means that the component's size is also greater than K. Several recent results indicate that searching a vertex of degree at least K in an heavy-tailed network takes $O(\log K)$ steps in average [1, 16], if the search is a depth-first search that always goes to the unvisited neighbour of highest degree. Thus, running an isolation test would be done in $O(\log K)$ average time instead of $O(K)$. Finally, the global complexity would become $O(m \log \log n)$ time.

5 Conclusion

Focusing on the speed-up method introduced by Gkantsidis et al. for the Markov chain Monte Carlo algorithm, we introduced a formal background allowing us to show that this heuristics is not optimal in its own family. We improved it in order to reach the optimal, and empirically confirmed the results.

Going further, we then took advantage of the characteristics of real-world networks to introduce an original method allowing the generation of random simple connected graphs with heavy-tailed degree distributions in $O(m \log n)$ time and $O(m)$ space. It outperforms the previous best known methods, and has the advantage of being extremly easy to implement. We also have pointed directions for further enhancements to reach a complexity of $O(m \log \log n)$ time. The empirical measurement of the performances of our methods show that it yields significant progress. We provide an implementation of this last algorithm [18].

Notice however that the last results rely on a conjecture, for which we have several arguments and strong empirical evidences, but were unable to prove.

References

1. Adamic, Lukose, Puniyani, and Huberman. Search in power-law networks. *Phys. Rev. E*, 64(046135), 2001.
2. Aiello, Chung, and Lu. A random graph model for massive graphs. *Proc. of the 32nd ACM STOC*, pages 171–180, 2000.
3. C. Berge. *Graphs and Hypergraphs*. North-Holland, 1973.
4. Erdos and Gallai. Graphs with prescribed degree of vertices. *Mat. Lapok*, 11:264–274, 1960.
5. Faloutsos, Faloutsos, and Faloutsos. On power-law relationships of the internet topology. *Proc. ACM SIGCOMM*, 29:251–262, 1999.
6. Gkantsidis, Mihail, and Zegura. The markov chain simulation method for generating connected power law random graphs. *in proc. ALENEX*, 2003.
7. S. L. Hakimi. On the realizability of a set of integers as degrees of the vertices of a linear graph. *SIAM Journal*, 10(3):496–506, 1962.
8. V. Havel. A remark on the existence of finite graphs. *Caposis Pest. Mat.*, 80:496–506, 1955.

9. Henzinger and King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. of ACM*, 46(4), 1999.
10. Holm, de Lichtenberg, and Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *STOC'98*, pages 79–89, 1998.
11. J. M. Roberts Jr. Simple methods for simulating sociomatrices with given marginal totals. *Social Networks*, 22:273–283, 2000.
12. Milo, Kashtan, Itzkovitz, Newman, and Alon. Uniform generation of random graphs with arbitrary degree seq. *sub. Phys. Rev. E*, 2001.
13. Molloy and Reed. A critical point for random graphs with a given degree sequence. *Random Struct. and Algo.*, pages 161–179, 1995.
14. Molloy and Reed. The size of the giant component of a random graph with a given degree sequence. *Comb., Prob. and Comp.*, 7:295, 1998.
15. R.Taylor. Constrained switchings in graphs. *Comb. Mat. 8*, 1980.
16. Sarshar, Boykin, and Roychowdhury. Scalable percolation search in power law networks. *P2P'04*, pages 2–9.
17. M. Thorup. Near-optimal fully-dynamic graph connectivity. *Proc. of the 32nd ACM STOC*, pages 343–350, 2000.
18. www.liafa.jussieu.fr/∼fabien/generation.

Randomized Quicksort and the Entropy of the Random Source*

Beatrice List, Markus Maucher, Uwe Schöning, and Rainer Schuler

Abt. Theoretische Informatik, Universität Ulm, 89069 Ulm, Germany

Abstract. The worst-case complexity of an implementation of Quicksort depends on the random source that is used to select the pivot elements. In this paper we estimate the expected number of comparisons of Quicksort as a function of the entropy of the random source. We give upper and lower bounds and show that the expected number of comparisons increases from $n \log n$ to n^2, if the entropy of the random source is bounded. As examples we show explicit bounds for distributions with bounded min-entropy and the geometrical distribution, as well as an upper bound when using a δ-random source.

Keywords: QuickSort, Randomized Algorithms, Entropy.

1 Introduction

Randomized QuickSort is the well known version of QuickSort [4] where the splitting element (the "pivot") is selected at random. It is known that the expected number of comparisons (for *every* input permutation of the array elements) is $(2 \ln 2) \cdot n \log_2 n - \Theta(n)$. Here, the expectation is taken over the random choices done in the algorithm. This analysis assumes random numbers which are independent and uniformly distributed.

We analyze randomized QuickSort without assuming such a "high entropy" of the underlying random source. Using a random number generator with a low entropy can result in a worst-case behavior that can go up to $\Theta(n^2)$.

Related work has been done by Karloff and Raghavan [5] (see also [12]) where the special case of a linear congruence generator is considered and a worst-case behavior of $\Omega(n^2)$ is shown.

Recursion for Expected Number of Comparisons

Let $T_\pi(n)$ be the expected number of comparisons done by randomized QuickSort, when operating on an input array $(a[1], \ldots, a[n])$ whose elements are permuted according to $\pi \in S_n$, that is,

$$a[\pi(1)] < a[\pi(2)] < \cdots < a[\pi(n)],$$

where S_n is the set of all permutations on $\{1, \ldots, n\}$.

* Work supported by DFG research grant Scho 302/6-1

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 450–460, 2005.

Let X be a random variable taking values between 1 and n (not necessarily under uniform distribution) which models the random number generator that is used to pick out a pivot element $a[X]$.

We obtain the following recursion for the expected complexity (i.e. number of comparisons) $T(n) = \max_{\pi \in S_n} T_\pi(n)$. We have $T(n) = 0$ for $n \leq 1$; and for $n > 1$ we get

$$T(n) = (n-1) + \sum_{i=1}^{n} p_i \cdot (T(i-1) + T(n-i)) \ .$$

That is, there are $n-1$ comparisons with the selected splitting element, and depending on the rank i of the pivot element within the array, there are $T(i-1)$ and $T(n-i)$ additional comparisons. Here p_i is the probability that the pivot element has rank i within the ordering of the array, that is, $p_i = Pr(\pi(X) = i)$. If the rank is not uniformly distributed among the numbers 1 to n, a worst case input permutation can be constructed such that the middle ranks receive relatively low probability and the extreme ranks (close to 1 or close to n) get relatively high probability, resulting in a large expected number of comparisons.

We give upper and lower bounds on the expected number $T(n)$ of comparisons. Lower bounds are given with respect to a fixed worst case input sequence (the already sorted list of elements).

We can show (see Theorem 1) that $T(n) \leq g(n) \cdot n \cdot \log_2 n$ for any function $g(n)$ greater than $1/\left(\min_\pi \sum_{i=1}^{n} p_i H\left(i/n\right)\right)$, where $H\left(i/n\right)$ is the binary entropy function. Note that $\min_\pi \sum_{i=1}^{n} p_i H\left(i/n\right)$ is independent of the permutation of the elements, i.e. is identical for all distributions p and q such that $p_i = q_{\pi(i)}$ for all i and some permutation π.

The lower bound (see Theorem 2) is derived for a fixed permutation (the sorted list of elements), where we can assume that the order is preserved in all recursive calls of QuickSort. Therefore the lower bound $T(n) \geq cng(n)$ (Theorem 2) is w.r.t. any function $g(n)$ less than $1/\sum_{i=1}^{n} p_i H\left(i/(n+1)\right)$, where p_i is the probability of selecting $a[i]$ as a pivot element.

2 Upper Bound on the Number of Expected Comparisons

Let (P_n) denote a sequence of probability distributions where $P_n = (p_{1,n}, \ldots, p_{n,n})$ is a distribution on $(1, \ldots, n)$. In the following we use p_i to denote $p_{i,n}$, since n is determined by the size of the array.

Theorem 1. *We have $T(n) \leq g(n) n \log_2 n$ for any monotone increasing function g with the property*

$$g(n) \ \geq \ \left(\min_{\pi \in S_n} \sum_{i=1}^{n} p_i \cdot H\left(\frac{i}{n}\right) \right)^{-1} \ ,$$

where $H(x) = -x \log_2 x - (1-x) \log_2 (1-x)$ is the binary entropy function.

Proof. By induction on n. Using the above recursion for $T(n)$ we obtain

$$\begin{aligned} T(n) &= (n-1) + \max_{\pi \in S_n} \sum_{i=1}^{n} p_i \cdot (T(i-1) + T(n-i)) \\ &\leq n + \max_{\pi} \sum_{i=1}^{n} p_i \cdot (g(i-1)(i-1)\log_2(i-1) + g(n-i)(n-i)\log_2(n-i)) \\ &\leq n + ng(n) \max_{\pi \in S_n} \sum_{i=1}^{n} p_i \cdot \left(\frac{i}{n} \log_2 i + \left(1 - \frac{i}{n}\right) \log_2(n-i) \right) \\ &= n + g(n) n \log_2 n - g(n) n \min_{\pi \in S_n} \sum_{i=1}^{n} p_i \cdot H\left(\frac{i}{n}\right) \quad . \end{aligned}$$

To finish the induction proof, this last expression should be at most $g(n)n\log_2 n$. This holds if and only if $g(n) \geq \left(\min_{\pi \in S_n} \sum_{i=1}^{n} p_i \cdot H\left(\frac{i}{n}\right) \right)^{-1}$ as claimed. □

Example: In the standard case of a uniform distribution $p_i = \frac{1}{n}$ we obtain $g(n) \geq \left(\frac{1}{n} \cdot \sum_{i=1}^{n} H\left(\frac{i}{n}\right)\right)^{-1}$. This is asymptotically $\left(\int_0^1 H(x)dx\right)^{-1} \approx 1.38$.

Another Example: In the median-of-three version of QuickSort (cf. [6, 10]), three different elements are picked uniformly at random and the median of the three is used as the pivot element. In this case $p_i = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$. Here the constant factor of the $n \log n$-term can be asymptotically estimated by

$$\left(6 \int_0^1 x(1-x)H(x)dx\right)^{-1} = \frac{12 \ln 2}{7} \approx 1.18 \quad .$$

Sorting the Probabilities

Using the symmetry of the function H around $\frac{1}{2}$ and its monotonicity, we get:

$$\min_{\pi \in S_n} \sum_{i=1}^{n} p_i \cdot H\left(\frac{i}{n}\right) \geq \min_{\pi \in S_n} \sum_{j=0}^{n-1} q_j \cdot H\left(\frac{j}{2n}\right) \quad .$$

Here, the q_j are a reordering of the p_i in the following way (assuming n is even):

$$\begin{aligned} q_0 = p_n,\ q_2 = p_{n-1},\ \ldots\ q_{n-2} &= p_{n/2+1} \\ q_1 = p_1,\ q_3 = p_2, \quad \ldots\ q_{n-1} &= p_{n/2} \end{aligned}$$

This new representation has the advantage that the H-values in the sum are in increasing order, and we can determine which permutation $\pi \in S_n$ actually achieves the minimum. Namely, the minimum is achieved if the q_j are ordered in decreasing order.

Lemma 1. *Given a sum of the form* $\sum_{j=1}^{n} a_j b_{\pi(j)}, \quad a_j, b_j \geq 0$, *where the* a_j *are sorted in strictly increasing order and a permutation* π, *the minimum value of the sum occurs when the permutation* π *is such that the* $b_{\pi(j)}$ *are sorted in decreasing order.*

Proof. Suppose that two elements b, b' are in the "wrong" order, i.e. $b < b'$. We compare the situation before and after exchanging b and b':

$$(a_i b + a_j b') - (a_i b' + a_j b) \;=\; (a_i - a_j)(b - b') \;<\; 0 \;\; .$$

□

3 A Lower Bound

To estimate a lower bound for the worst-case running time of QuickSort, we consider as input the already sorted array of numbers. Further we assume that the partitioning step of QuickSort leaves the elements of the two sub-arrays in the same relative order as in the input array.

Recall that pivot-elements are chosen according to a sequence of probability distributions (P_i), where distribution P_i defines the probabilities on arrays of size i, i.e. $P_i = (p_{i,1}, \ldots, p_{i,i})$. Note that if the $p_{i,j}$ are sorted in decreasing order, then a worst-case input is the already sorted sequence of numbers. In fact, if the sequence of probability distributions (P_i) is sufficiently *uniform*, it should be possible to construct a worst-case input by sorting probabilities as described in Section 2.

Theorem 2. *(i) For any sequence of probability distributions* (P_i) *it holds that* $T(n) \geq c \cdot g(n) \cdot n - n$, *for some constants* $c > 0$ *and* n_0, *if for all* $n > n_0$, g *satisfies the two conditions*

$$g(n) \leq \left(\sum_{i=1}^{n} p_{i,n} \left(1 - \frac{(i-1)^2}{n^2} - \frac{(n-i)^2}{n^2} \right) \right)^{-1}$$

$$\textit{and} \qquad \frac{g(i)}{g(n)} \geq \frac{i}{n} \quad \textit{for all} \quad 0 \leq i \leq n.$$

(ii) Furthermore, Part (i) still holds if we replace the two conditions by

$$g(n) \leq \left(\sum_{i=1}^{n} p_{i,n} H\left(\tfrac{i}{n+1} \right) \right)^{-1} \textit{ and } \tfrac{g(i)}{g(n)} \geq \tfrac{i}{n} \quad \textit{for} \quad 0 \leq i \leq n.$$

Proof. We prove (i) first, by induction. For $n \leq n_0$, just set the constant $c \leq 1$ small enough.

Now we look at the case $n > n_0$. Let $P = (p_1, \ldots, p_n)$ be a distribution where p_i is the probability that we choose as a pivot element the element with rank i. Using the induction hypothesis, it holds that

$$T(i-1)+T(n-i) \geq c\cdot(i-1)\cdot g(i-1)+c\cdot(n-i)\cdot g(n-i)-(n-1)$$
$$\geq c\cdot n\cdot g(n)\cdot\left(\frac{(i-1)^2}{n^2}+\frac{(n-i)^2}{n^2}\right)-(n-1)$$
$$= c\cdot n\cdot g(n)-c\cdot n\cdot g(n)\cdot\left(1-\frac{(i-1)^2}{n^2}-\frac{(n-i)^2}{n^2}\right)-(n-1)\ .$$

Therefore, $T(n)=n-1+\sum_{i=1}^{n} p_i(T(i-1)+T(n-i))$

$\geq c\cdot n\cdot g(n)-c\cdot n\cdot g(n)\cdot\sum_{i=1}^{n} p_i\left(1-\frac{(i-1)^2}{n^2}-\frac{(n-i)^2}{n^2}\right)$.

As $c\leq 1$, we can finish the induction if $g(n)\leq\left(\sum_{i=1}^{n} p_i\left(1-\frac{(i-1)^2}{n^2}-\frac{(n-i)^2}{n^2}\right)\right)^{-1}$.

The proof of part (ii) is quite similar: For $n\geq n_0$,

$$T(i-1)+T(n-i)\geq cng(n)\cdot\left(\frac{(i-1)^2}{n^2}+\frac{(n-i)^2}{n^2}\right)-(n-1)$$
$$= cng(n)\cdot\left(\frac{(i-1)^2}{n^2}+\frac{(n-i)^2}{n^2}+H\left(\frac{i}{n+1}\right)\right)-ng(n)\cdot H\left(\frac{i}{n+1}\right)-(n-1)$$
$$\geq cng(n)-cng(n)\cdot H\left(\frac{i}{n+1}\right)-(n-1).$$

The last inequality uses the fact that for integers $n\geq 1$ and i with $0\leq i\leq n$, $\frac{(i-1)^2}{n^2}+\frac{(n-i)^2}{n^2}+H\left(\frac{i}{n+1}\right)\geq 1$ (which can be shown quite easily). Now $T(n)=n-1+c\cdot\sum_{i=1}^{n} p_i(T(i-1)+T(n-i))\geq cng(n)-cng(n)\cdot\sum_{i=1}^{n} p_i H\left(\frac{i}{n+1}\right)$.

Again using $c\leq 1$, we can finish the induction if $g(n)\leq\left(\sum_{i=1}^{n} p_i H\left(\frac{i}{n+1}\right)\right)^{-1}$. □

In the second part of Theorem 2 the lower bound is given using the entropy function, similar to the upper bound in Theorem 1. This shows that up to a logarithmic factor we yield matching upper and lower bounds.

4 Distributions with Bounded Entropy

The uniform distribution on $[1,n]=\{1,\ldots,n\}$ has maximal entropy. In this section we consider distributions which have bounded entropy.

Uniform Distributions on a Subset of $\{1,\ldots,n\}$

First we consider distributions with positive probability on subsets of $[1,n]$. Let $t(n)=o(n)$ be a time constructible monotone (increasing) function. Define a distribution $P=(p_1,\ldots,p_n)$ such that

$$p_i = \begin{cases} 1/t(n), & \text{if rank } a_i \le t(n)/2 \text{ or rank } a_i > n - t(n)/2 \\ 0, & \text{otherwise} \end{cases}$$

That is, we choose the pivot element randomly using a uniform distribution among only the worst $t(n)$ array elements.
Now $\sum_{i=1}^{n} p_i H\left(i/(n+1)\right)$ resp. $\sum_{i=1}^{n} p_i \cdot H(i/n)$ are bounded as follows:

$$\sum_{i=1}^{n} p_i H\left(\frac{i}{n+1}\right) \le \frac{t(n)}{2n} \log\left(n+1\right), \qquad \sum_{i=1}^{n} p_i H\left(\frac{i}{n}\right) \ge \frac{t(n)}{4n} \log\left(\frac{2n}{t(n)}\right) .$$

This gives $T(n) \le n \log(n) \cdot \frac{4n}{t(n)}$ as an upper bound and $T(n) \ge \frac{cn^2}{t(n) \log n} - n$ as a lower bound, for some constant c.

Proof. An upper bound $T(n) \le g(n) \cdot n \log_2 n$ can be estimated as follows.

$$\begin{aligned} \sum_{i=1}^{n} p_i \cdot H\left(\frac{i}{n}\right) &= 2 \sum_{i=1}^{t(n)/2} \frac{1}{t(n)} \cdot H\left(\frac{i}{n}\right) = \frac{2}{t(n)} \sum_{i=1}^{t(n)/2} H\left(\frac{i}{n}\right) \\ &= \frac{2}{t(n)} \sum_{i=1}^{t(n)/2} - \left(\frac{i}{n} \log\left(\frac{i}{n}\right) + \frac{n-i}{n} \log\left(\frac{n-i}{n}\right)\right) \\ &\ge \frac{2}{t(n)} \sum_{i=1}^{t(n)/2} \frac{i}{n} \log\left(\frac{n}{i}\right) \ge \frac{2}{n \cdot t(n)} \log\left(\frac{2n}{t(n)}\right) \sum_{i=1}^{t(n)/2} i \\ &\ge \frac{2}{n \cdot t(n)} \log\left(\frac{2n}{t(n)}\right) \frac{(t(n)/2) \cdot (t(n)/2+1)}{2} \ge \frac{t(n)}{4n} \log\left(\frac{2n}{t(n)}\right). \end{aligned}$$

With $g(n) = \frac{4n}{t(n) \log(2n/t(n))}$, it follows that $T(n) \le \frac{4n^2}{t(n)} \cdot \frac{\log_2 n}{\log_2(2n/t(n))}$ (see Theorem 1). In the same way the lower bound can be calculated:

$$\begin{aligned} &\sum_{i=1}^{n} p_i \cdot H\left(\frac{i}{n+1}\right) = 2 \sum_{i=1}^{t(n)/2} \frac{1}{t(n)} \cdot H\left(\frac{i}{n+1}\right) \le \ldots \le \\ &\le \frac{t(n)+1}{2(n+1)} \left(\log\left(n+1\right) - \log t(n) + 2\right)) \le \frac{t(n)+2}{2(n+1)} \log\left(\frac{4(n+1)}{t(n)}\right) \end{aligned}$$

where we use that $\sum_{i=1}^{t(n)/2} i \log i \le \sum_{i=1}^{t(n)/2} i(\log(t(n)/2) - 1)$ (which can be shown by induction).
With the function $g(n) = \frac{2(n+1)}{(t(n)+1) \log\left(\frac{4(n+1)}{t(n)}\right)}$, we receive a lower bound of

$$T(n) \ge \frac{2cn(n+1)}{(t(n)+1) \log\left(\frac{4(n+1)}{t(n)}\right)} - n = \Omega\left(\frac{n^2}{t(n) \log\left(\frac{4n}{t(n)}\right)} - n\right) . \qquad \square$$

Min-entropy

Uniform distributions on subsets of $[1, \ldots, n]$ are just a special case of distributions with bounded *min-entropy*.

Definition 1. *A distribution* $(p_1, \ldots, p_n)$ *has* min-entropy k *if* $\max_i p_i = 2^{-k}$ *(cf. [8]).*

Let $P = (p_1, \ldots, p_n)$ be a distribution with min-entropy k. Then we get an upper bound of $T(n) \leq \frac{4n^2}{2^k}$ and a lower bound of $T(n) \geq \frac{cn^2}{2^k \log n} - n$, for $c > 0$.

Proof. $\displaystyle\sum_{i=1}^{n} p_i \cdot H(i/n) \geq 2 \sum_{i=1}^{2^k/2} \frac{1}{2^k} \cdot H(i/n) \geq \frac{2^k}{4n} \log\left(\frac{2n}{2^k}\right)$, and

$$\sum_{i=1}^{n} p_i \cdot H\left(\frac{i}{n+1}\right) \leq 2 \sum_{i=1}^{2^k/2} \frac{1}{2^k} \cdot H\left(\frac{i}{n+1}\right) \leq \frac{2^k+1}{2(n+1)} \log\left(\frac{2(n+1)}{2^k}\right) \quad ,$$

and thus $T(n) \leq \dfrac{4n^2}{2^k} \cdot \dfrac{\log_2 n}{\log_2(2n/2^k)}$ and $T(n) \geq \dfrac{2cn(n+1)}{(2^k+1)\log\left(\frac{2(n+1)}{2^k}\right)} - n.$ □

So, for min-entropy 0 (this includes the deterministic case) we get

$$T(n) \leq \frac{4n^2}{1} \cdot \frac{\log_2 n}{\log_2(2n)} = 4n^2 \frac{\log_2 n}{\log_2 n + 1} \leq 4n^2 \qquad \text{and}$$

$$T(n) \geq \frac{cn(n+1)}{\log(2(n+1))} - n \geq \frac{cn^2}{\log(n+1)+1} - n = \theta\left(\frac{n^2}{\log n}\right) \quad ,$$

and for min-entropy $\log_2 n$ (all pivot elements are equally distributed), we have

$$T(n) \leq \frac{4n^2}{n} \cdot \frac{\log_2 n}{\log_2 2} = 4n \log_2 n \ .$$

Bounds for Geometric Distributions

We consider the case that pivot elements are selected using a geometric distribution. The probability of picking an element with rank i as pivot is given by $p_i = q^{i-1}(1-q)$. More generally, we allow the geometric distribution to depend on the size n of the array, i.e., we define (P_i) using $q := 1 - \frac{1}{f(i)}$ for some (time constructible monotone) function $f = o(n)$. An additional probability of q^n is assigned to the best resp. worst pivot element (depending on if we consider a lower or upper bound), so that all p_i sum up to 1.

To estimate a lower bound on the number of comparisons, we use Theorem 2 and estimate $\sum_{i=1}^{n} p_i \left(1 - \frac{(i-1)^2}{n^2} - \frac{(n-i)^2}{n^2}\right) \leq \frac{cf(n)}{n}$, for a constant c.

Proof. Using the fact that $q^i = \left(1-\frac{1}{f(n)}\right)^i = \left(1-\frac{1}{f(n)}\right)^{f(n)\cdot\frac{i}{f(n)}} \le e^{-\frac{i}{f(n)}}$, it follows that

$$\begin{aligned}
&\sum_{i=1}^{n} p_i \left(1-\frac{(i-1)^2}{n^2}-\frac{(n-i)^2}{n^2}\right)\\
&\le q^n\left(1-\frac{(\frac{n}{2}-1)^2}{n^2}-\frac{(\frac{n}{2})^2}{n^2}\right)+\frac{1}{q}\sum_{i=1}^{n} q^i(1-q)\left(1-\frac{(i-1)^2}{n^2}-\frac{(n-i)^2}{n^2}\right)\\
&\le q^n+\frac{1}{qn^2}\sum_{i=1}^{n} q^i(1-q)\,(2ni+2i)\\
&=\left(1-\frac{1}{f(n)}\right)^n+\frac{(2n+2)f(n)}{(f(n)-1)n^2}\sum_{i=1}^{n}\left(1-\frac{1}{f(n)}\right)^i\frac{i}{f(n)}\ .
\end{aligned}$$

We split the sum and see that for $k=0,1,2,\ldots$

$$\begin{aligned}
&\sum_{i=kf(n)+1}^{(k+1)f(n)}\left(1-\frac{1}{f(n)}\right)^i\frac{i}{f(n)}\sum_{i=kf(n)+1}^{(k+1)f(n)} e^{-\frac{i}{f(n)}+\ln\frac{i}{f(n)}}=\sum_{j=1}^{f(n)} e^{-\frac{kf(n)+j}{f(n)}+\ln\frac{kf(n)+j}{f(n)}}\\
&\le\sum_{j=1}^{f(n)} e^{-k-\frac{j}{f(n)}+\ln(k+1)}=e^{-k+\ln(k+1)}\sum_{j=1}^{f(n)} e^{-\frac{j}{f(n)}}\le e^{-k+\ln(k+1)}\cdot f(n)\ .
\end{aligned}$$

Then we get

$$\begin{aligned}
&\left(1-\frac{1}{f(n)}\right)^n+\frac{(2n+2)f(n)}{n^2(f(n)-1)}\sum_{i=1}^{n}\left(1-\frac{1}{f(n)}\right)^i\frac{i}{f(n)}\\
&=\left(1-\frac{1}{f(n)}\right)^n+\frac{(2n+2)f(n)}{n^2(f(n)-1)}\sum_{k=0}^{\lceil n/f(n)\rceil}\ \sum_{i=kf(n)+1}^{(k+1)f(n)}\left(1+\frac{1}{f(n)}\right)^i\cdot\frac{i}{f(n)}\\
&\le e^{-\frac{n}{f(n)}}+\frac{(2n+2)f(n)^2}{n^2(f(n)-1)}\sum_{k=0}^{\infty}\frac{k+1}{e^k}\le\frac{cf(n)}{n}\ \text{ for a constant } c.
\end{aligned}$$

For the last inequality, note that $f(n)=o(n)$, so that $e^{-\frac{n}{f(n)}}=o\left(\frac{f(n)}{n}\right)$.

Using Theorem 2, we get a lower bound of $c'n^2/f(n)$ for the running time of the QuickSort algorithm, for some constant c'. □

To get an upper bound for geometric distributions we estimate similarly

$$\sum_{i=1}^{n} p_i H\left(\frac{i}{n}\right)\ge\frac{f(n)}{n}\left(1-e^{-\frac{n}{2f(n)}}\cdot\frac{n}{f(n)}\right)\ ,$$

which gives $T(n)\le\frac{cn^2\log n}{f(n)}$ as upper bound, for some $c>0$. (Proof omitted)

5 The δ-Random Source

A general model of a random bit source is the δ-random-source. Since the bias of each bit is a function of the previous output, it can be applied as an adversary argument and is particularly suited for worst-case analysis. See also [1, 9, 11].

Definition 2 (See [1]). *A δ-random-source is a random bit generator. Its bias may depend on the bits it has previously output, but the probability to output "1" must be in the range $[\delta, 1-\delta]$. Therefore, it has an internal state $\omega \in \{0,1\}^*$, denoting its previously output bits.*

To obtain a random number X in the range $1, \ldots, n$ from the δ-random-source, we output $\lceil \log n \rceil$ bits and interpret them as a number Y. Then, we set $X := (Y \bmod n) + 1$.

Lemma 2 (See [2]). *For each p with $0 < p < \frac{1}{2}$, there exists a constant c, such that for all $n \in \mathbb{N}$: $c(p) \cdot \dfrac{2^{H(p)\cdot n}}{\sqrt{n}} \leq \displaystyle\sum_{j=0}^{\lfloor np \rfloor} \binom{n}{j} \leq 2^{H(p)\cdot n}$.*

Theorem 3. *For each δ-random-source, $0 < \delta < \frac{1}{2}$, there exists $n_0 \in \mathbb{N}$, such that for each $n > n_0$, and each permutation π, Theorem 1 can be applied with $g(n) = c(\delta) \cdot \dfrac{1}{\sqrt{\log n}} \cdot n^{1-H(\delta)}$, where the random bits are produced by a δ-random-source and $c(\delta)$ is a constant that depends on δ.*

Proof. From the symmetry and monotony of the entropy function it follows that for each s

$$\sum_{i=1}^{n} p_i \cdot H\left(\frac{i}{n}\right) \geq \left(1 - \sup_{\pi,\tilde{\omega}} \sum_{j=1}^{s-1} p_j\right) \cdot H\left(\frac{s}{2n}\right), \tag{1}$$

where p_j depends on π and on the internal state $\tilde{\omega}$ of the random source.

Now we examine the two factors on the righthand side of (1) separately. We set $k := \lceil \log n \rceil$ and $s := \frac{1}{2} \sum_{j=0}^{\lfloor \delta k \rfloor} \binom{k}{j}$. Since

$$p_j = \begin{cases} Pr[Y = \pi(j)], & n + \pi(j) \geq 2^k \\ Pr[Y = \pi(j)] + Pr[Y = \pi(j) + n] & \text{otherwise} \end{cases},$$

we get for the first factor of (1)

$$\sup_{\pi,\tilde{\omega}} \sum_{j=1}^{s-1} p_j \leq \sup_{\tilde{\omega}} \max_{M \subseteq \{0,1\}^k, |M| = 2s} Pr[Y \in M] \leq \sum_{j=0}^{\lfloor \delta k \rfloor} \binom{k}{j} \delta^j (1-\delta)^{k-j} .$$

Here we use the result from [1], that the maximum probability of hitting a set of a certain size can be achieved by an "extreme" δ-random-source that always outputs "0" with probability δ.

Since $\lim_{k\to\infty} \sum_{j=0}^{\lfloor \delta k \rfloor} \binom{k}{j} \delta^j (1-\delta)^{k-j} = \frac{1}{2}$, (which follows from the DeMoivre-Laplace Limit Theorem,) there exists a constant $c'(\delta)$, so that $\sup_{\pi,\tilde{\omega}} \sum_{j=1}^{s-1} p_j \leq c'(\delta)$.

Now we consider the second factor of (1). We use the monotony of $H(x)$ on the intervall $[0, \frac{1}{2}]$ and Lemma 2:

$$H\left(\frac{s}{2n}\right) \geq H\left(\frac{s}{2^{k+1}}\right) \geq H\left(c_1(\delta) \cdot \frac{2^{(H(\delta)-1)k}}{4\sqrt{k}}\right) .$$

We consider $\delta < \frac{1}{2}$ (so that $H(\delta) < 1$) and use that $H(x) \geq -x \log x$ to get

$$H\left(\frac{s}{2n}\right) \geq c_1(\delta) \cdot \frac{2^{(H(\delta)-1)k}}{4\sqrt{k}} \cdot \left[(1 - H(\delta))k - \log \frac{c_1(\delta)}{4\sqrt{k}}\right].$$

For k big enough ($k > k_0$ corresponds to $n > n_0$), there is a constant $c''(\delta)$ so that

$$H\left(\frac{s}{2n}\right) \geq c''(\delta) \cdot \sqrt{k} \cdot 2^{(H(\delta)-1)k}.$$

Combining the results, there is a $n_0 \in \mathbb{N}$ and a $c^*(\delta)$, such that for all $n \geq n_0$, and all permutations π on $\{0, \ldots, n-i\}$ and all states $\tilde{\omega} \in \{0,1\}^*$ of the generator the following holds:

$$\sum_{i=1}^{n} p_i \cdot H\left(\frac{i}{n}\right) \geq c^*(\delta) \cdot \sqrt{\lceil \log n \rceil} \cdot 2^{(H(\delta)-1)\lceil \log n \rceil} \geq \frac{1}{c(\delta)} \cdot \sqrt{\log n} \cdot n^{H(\delta)-1} ,$$

which leads to the expected running time of $T(n) \leq c(\delta) \cdot n^{2-H(\delta)} \cdot \sqrt{\log n}$. □

References

1. Alon, N. and Rabin, M. O.: Biased coins and randomized algorithms. In Preparata, F.P., Micali, S., eds.: Advances in Computing Research 5, JAI Press (1989) 499–507
2. Ash, R.B.: Information Theory. Dover (1965)
3. Devroye, L.: On the probabilistic worst-case time of "FIND". Algorithmica 31 (2001) 291–303
4. Hoare, C.A.R.: Quicksort. Computer Journal 5(1) (1962) 10–15
5. Karloff, H.J. and Raghavan, P.: Randomized algorithms and pseudorandom numbers. Journal of the Association for Computing Machinery 40 (1993) 454–476
6. Knuth, D.: The Art of Computer Programming. Vol 3: Sorting and Searching, Addison-Wesley (1973)
7. List, B.: Probabilistische Algorithmen und schlechte Zufallszahlen. PhD thesis, Universität Ulm(1999)
8. Luby, M.: Pseudorandomness and Cryptographic Applications. Princeton University Press (1996)
9. Papadimitriou, C. H.: Computational Complexity. Addison-Wesley (1994)

10. Robert Sedgewick, R., Flajolet, P.: Analysis of Algorithms. Addison-Wesley(1994)
11. Santha, M., Vazirani, U.V.: Generating quasi-random sequences from slightly random sources. Proceedings of the 25th IEEE (1984)
12. Tompa, M.: Probabilistic Algorithms and Pseudorandom Generators. Lecture Notes (1991)

Subquadratic Algorithm for Dynamic Shortest Distances*

Extended Abstract

Piotr Sankowski

Institute of Informatics, Warsaw University
ul. Banacha 2, 02-097 Warsaw
sank@mimuw.edu.pl

Abstract. In this paper we extend a technique introduced in [14] for dynamic matrix functions. We present dynamic algorithms for computing matrix determinant and matrix adjoint over commutative rings. These algorithms are then used to construct an algorithm for dynamic shortest distances in unweighted graph. Our algorithm supports updates in $O(n^{1.932})$ randomized time and queries in $O(n^{1.288})$ randomized time. These bound improve over the previous results and solve a long-standing open problem if sub-quadratic dynamic algorithms exist for computing all pairs shortest distances.

In this paper we consider dynamic evaluation of algebraic functions over commutative rings. We show how to compute determinant and matrix adjoint dynamically. Let $\mathcal{R} = (R, +, \cdot, 0, 1)$ be a commutative ring with elements from set R and appropriately defined addition $+$ and multiplication $\cdot$. Let $f : R^n \to R^m$ be an algebraic function over the ring. Given an initial input vector $(x_1, x_2, \ldots, x_n) \in R^n$, a dynamic algorithm is allowed some preprocessing and then must quickly handle the following requests:

- **update**$(k,\ x'_k)$ change the k'th input value to x'_k,
- **query**(k) return the k'th output value.

We consider the following two dynamic matrix problems.

- **determinant** $R^{n^2} \to R$: The input is interpreted as $n \times n$ matrix over the ring $\mathcal{R}$. The output is its determinant.
- **adjoint** $R^{n^2} \to R^{n^2}$: The input is interpreted as $n \times n$ matrix A. The output is interpreted as $n \times n$ adjoint of the input matrix, i.e., $\mathrm{adj}(A)_{i,j} = (-1)^{i+j} \det(A^{j,i})$, where $A^{j,i}$ denotes the $(n-1) \times (n-1)$ matrix resulting from the deletion of the j'th row and the i'th column of A.

We construct algorithms for the above problems in the case of two types of updates: *row and column updates*, i.e., we change i'th row or column of the matrix to a given vector; *simple updates*, i.e., changes of only one entry of the matrix.

* Research supported by KBN grant 4T11C04425

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 461–470, 2005.

The first paper concerning dynamic evaluation of algebraic function over matrices such as matrix determinant and matrix inverse, was written by Rief and Tate [13]. Two years later Frendsen, Hansen and Miltersen [7] presented the lower bounds of $\Omega(n)$ for algebraic matrix problems. In [14] the first dynamic algorithms were shown for the problem of computing matrix determinant, matrix adjoint, matrix inverse and solving linear system of equation. These algorithms assumed that matrices are defined over a field. In this paper we extend these results to commutative rings. It is also possible to extend these results to Euclidean rings, where due to some notion of divisibility, the obtained algorithms work faster than in the case of general rings.

The dynamic matrix algorithms were used in [14] as a black box in construction of dynamic algorithms for transitive closure. The algorithms developed in this paper will be used to a solve dynamic all pairs shortest path problem.

A dynamic graph algorithm maintains information about a given property P of a graph subject to dynamic changes. We maintain a directed graph $G = (V, E)$ under an intermixed sequence of edge updates, i.e., operations **insert(**e**)** that insert e into edge set E and operations **delete(**e**)** that delete e from edge set E.

The first fully dynamic algorithm for general graphs with integer weights was presented by King [11]. The running time of the algorithm is $O(n^{2.5}\sqrt{C \log n})$ per update. Later similar results, but in the case of real edge weights, were obtained by Demetrescu and Italiano in [4, 5]. They assumed that there are at most S different real edge weights and obtained an algorithm supporting updates in $O(n^{2.5}\sqrt{S \log^3 n})$ time and queries in constant time. They also presented two families of trade-off algorithms that have smaller update time but at a cost of bigger query time. The final step in obtaining $\tilde{O}(n^2)$ [1] update time was made by Demetrescu and Italiano in [6].

In this paper we show an algorithm for maintaining dynamically the lengths of the shortest paths in unweighted digraphs. Notice, that if the algorithm answers queries in $O(1)$ time it has to maintain the distance matrix explicitly. Since an update may change $\Omega(n^2)$ entries of the matrix, the bound of $\Theta(n^2)$ worst-case time seems to be the best we can hope for. Thus it is interesting to know if allowing greater query time, one can reduce the update time below $O(n^2)$. Till now no such algorithm has been known.

The applications of the matrix algorithms to graphs are based on a construction of a appropriate adjacency matrix of the graph, then the edge updates are then translated into simple updates of the matrix. Using an algorithm for dynamic matrix inverse over formal power series we develop an algorithm for computing lengths of the shortest paths that use at most k edges. This algorithm combined with the standard technique of path decomposition gives an algorithm supporting updates in $O(n^{1.932})$ and queries in $O(n^{1.288})$ time. This result resolves the open question (see e.g. [4–6]) whether algorithms with subquadratic update and query time exist. The problem of dynamic single source shortest distances seems inherently simpler than dynamic all pair shortest distances, but till now the best solution for this problem was evaluating everything

[1] Throughout the paper, we use $\tilde{O}(f(n))$ to to denote $O(f(n)\text{polylog(n)})$.

from the scratch, and this takes $O(n^2)$ time. Thus our result also resolves the question if more efficient algorithms for this problem exists. However, our result is only of a theoretical importance, the $\tilde{O}(n^2)$ algorithm from [6] is surely practically more efficient.

1 Fast Multiplication of Rectangular Matrices

Let us denote by $\omega(1,\epsilon,1)$ the exponent of multiplication of an $n \times n^\epsilon$ matrix by an $n^\epsilon \times n$ matrix. For $\epsilon = 1$ we get the exponent of square matrix multiplication $\omega = \omega(1,1,1)$. The best bound on ω is currently $\omega < 2.376$ [3]. Coppersmith [2] showed that it is possible to multiply an $n \times n^{0.294}$ matrix by an $n^{0.294} \times n$ matrix in $\tilde{O}(n^2)$ arithmetic operations. Let $\alpha = \sup\{\epsilon : \omega(1,\epsilon,1) = 2 + o(1)\}$, so $\alpha \geq 0.294$. Combining bounds on α and ω Huang and Pan [10] showed the following lemma.

Lemma 1 (Huang and Pan [10]). *Let* $\omega = \omega(1,1,1) < 2.376$ *and let* $\alpha = \sup\{\epsilon : \omega(1,\epsilon,1) = 2 + o(1)\} \geq 0.294$. *Then*

$$w(1,\epsilon,1) \leq \begin{cases} 2 + o(1) & \text{if } \ 0 \leq \epsilon \leq \alpha, \\ 2 + \frac{\omega-2}{1-\alpha}(\epsilon - \alpha) + o(1) & \text{if } \ \alpha \leq \epsilon \leq 1. \end{cases}$$

We also use the fact shown by Bunch and Hopcroft [1] that matrix inverse can be computed in the matrix multiplication time.

2 Dynamic Matrix Problems: Division-Free Algorithms

In order to construct division free versions of the algorithms presented in [14] we can use the standard technique introduced by Strassen [17]. He showed how to compute the determinant of a matrix without divisions in $\tilde{O}(n^{\omega+1})$ ring operations. In our case this construction has the same impact on the complexity, i.e., it adds 1 to the exponent. The idea is to work in $\mathcal{R}[[u]]$ — the ring of formal power series over $\mathcal{R}$. Let A be a matrix over the ring $\mathcal{R}$, we define $A(u) = I + u(A - I)$. Thus we have $A = A(u)|_{u=1}$, $\det(A) = \det(A(u))|_{u=1}$ and $\mathrm{adj}(A) = \mathrm{adj}(A(u))|_{u=1}$. The determinant of the matrix $A(u)$ can be computed with the standard Gaussian Elimination, because during elimination the elements on the diagonal are always invertible. They are of the form $1 - z$, where $z \in u\mathcal{R}[[u]]$. We have

$$\frac{1}{1-z} = 1 + z + z^2 + \ldots = (1+z)(1+z^2)(1+z^4)\ldots, \tag{1}$$

and thus it is possible to compute this quantity without inverting elements of $\mathcal{R}$. Let $R^n[[u]]$ denote the ring of the formal power series modulo u^{n+1}. The result of the evaluation $\det(A(u))$ – is a polynomial of degree n in u. Notice that terms in the result of the operations in $R[[u]]$ do not depend on higher degree terms in the arguments, so the whole computation can be carried in $R^n[[u]]$. Assuming that the elements of $R^n[[u]]$ can be multiplied with use of $O(n\log(n)\log(\log(n)))$ operations in R [12, 15] we obtain a complexity of $\tilde{O}(n^{\omega+1})$ for computing the determinant without divisions.

2.1 Dynamic Determinant: Row and Column Updates

In this section we present an algorithm supporting updates in $O(n^3)$ ring operations. We first recall the known fact that the matrix $A(u)$ is non-singular for all matrices A:

$$A(u)^{-1} = \frac{1}{I + u(A - I)} = \sum_{i=0}^{\infty} (-u(A - I))^i, \tag{2}$$

To verify the above equation it is sufficient multiply it by $A(u)$.

In the algorithm we maintain the inverse of the matrix $A(u)$ and recompute it after the change in A. Let us consider an update of the i'th column of A to the vector v. After the update we get matrix $A' = A + (v - (A)_i)e_i^T$, where $(A)_i$ is the i'th column of A and e_i is the vector with 1 on the i'th place and 0's on the other places. In order to compute the inverse of the matrix after the update we proceed as follows. First we compute matrix B such that:

$$A'(u) = I + u\left(A + (v - (A)_i)e_i^T - I\right) = A(u) \cdot B. \tag{3}$$

In the case of an update of the i'th row to v we proceed similarly as in the case of the column update but now:

$$A'(u) = I + u\left(A + e_i(v^T - (A)^i) - I\right) = B \cdot A(u), \tag{4}$$

where $(A)^i$ is the i'th row of matrix A.

Let us substitute $B = I + B'$. Then:

$$I + u\left(A + (v - (A)_i)e_i^T - I\right) = A(u) \cdot (I + B'),$$

$$A(u) + u(v - (A)_i)e_i^T = A(u) \cdot (I + B'),$$

$$u(v - (A)_i)e_i^T = A(u) \cdot B'.$$

As we have shown above $A(u)$ is invertible, so we have:

$$B' = A(u)^{-1}u(v - (A)_i)e_i^T$$

The matrix $(v-(A)_i)e_i^T$ has non-zero elements only in the i'th column, so B' has non-zero elements only in the i'th column as well and we can write $B' = ube_i^T$, and finally we get:

$$b = A(u)^{-1}(v - (A)_i), \tag{5}$$

$$B = I + B' = I + ube_i^T = I + uA(u)^{-1}(v - (A)_i)e_i^T.$$

Similarly as in (2) we can show that the matrix B is invertible.

$$B^{-1} = \frac{1}{I + ube_i^T} = \sum_{k=0}^{\infty} \left(-ube_i^T\right)^k = \sum_{k=0}^{\infty} \left(-uA(u)^{-1}(v - (A)_i)e_i^T\right)^k. \tag{6}$$

The vector b can be computed in $O(n^2)$ operations in $\mathcal{R}^n[[u]]$. Having b we can compute B^{-1} also in $O(n^2)$ operations. We have $A'^{-1} = B^{-1}A^{-1}$. Because the matrix B^{-1} has only $O(n)$ non-zero entries, this multiplication can be done with $O(n^2)$ multiplications in $\mathcal{R}^n[[u]]$. Thus for the update we need $\tilde{O}(n^3)$ operations in $\mathcal{R}$.

Theorem 1. *The problems of dynamic determinant and matrix adjoint over the commutative ring $\mathcal{R}$, with row and column updates, can be solved with the following costs:*

- **initialization** $\tilde{O}(n^{\omega+1})$ *ring operations,*
- **update** $\tilde{O}(n^3)$ *ring operations (worst-case),*
- **query** $O(1)$ *ring operations (worst-case).*

Proof. In the above construction we have shown how to maintain the inverse of the matrix $A(u)$. Notice that $\det(A'(u)) = (1 + b_i)\det(A(u))$ and the determinant of $A(u)$ can also be maintained during updates. The adjoint of the matrix $A(u)$ is given by $\mathrm{adj}(A(u))_{ij} = \det(A(u))(A(u)^{-1})_{ij}$. In order to be able to answer queries in constant number of operations we have to recompute the matrix $\mathrm{adj}(A) = \mathrm{adj}(A(u))|_{u=1}$ after every update. This can be done with use of $\tilde{O}(n^3)$ ring operations.

2.2 Dynamic Determinant: Simple Updates

The proof of Theorem 1 is similar to the proof of the theorems presented in [14]. The only difference is that we perform computations over $\mathcal{R}^n[[u]]$ and so we have to prove that all inversions we need are possible, (2) and (6). The following theorem is taken from [14].

Theorem 2. *[Sankowski [14]] The problems of dynamic determinant and matrix adjoint over a field, with non-singular simple updates can be solved with the following costs:*

- **initialization** $O(n^{\omega})$ *arithmetic operations,*
- **update** $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$ *arithmetic operations (worst-case),*
- **query for adjoint** $O(n^{\epsilon})$ *arithmetic operations (worst-case),*
- **query for determinant** $O(1)$ *arithmetic operations (worst-case).*

In order to prove Theorem 2 the author constructed a lazy computations scheme from similar algorithm as in Theorem 1. This lazy computation schemes can be also applied in the case of Theorem 1 as a result we get the following.

Theorem 3. *The problems of dynamic determinant and matrix adjoint over the commutative ring $\mathcal{R}$, with simple updates, can be solved with the following costs:*

- **initialization** $\tilde{O}(n^{\omega+1})$ *ring operations,*
- **update** $\tilde{O}(n^{1+\omega(1,\epsilon,1)-\epsilon} + n^{2+\epsilon})$ *ring operations (worst-case),*
- **query for the adjoint** $\tilde{O}(n^{1+\epsilon})$ *ring operations (worst-case),*
- **query for determinant** $O(1)$ *ring operations (worst-case).*

3 Maintaining a Part of the Inverse

In this section we show how the algorithm from Theorem 2 can be used to maintain only the part of the inverse of adjoint matrix. We will use this property in Section 4 in the construction of dynamic algorithms for the distance problem. It is possible to maintain in subquadratic time a part of the inverse matrix given by a subset X of the rows and a subset Y of the columns, i.e., when we are only allowed to query the matrix $(A^{-1})_{X,Y}$, where $A_{X,Y}$ is the submatrix of A corresponding to rows from the set X and columns from the set Y. If $|X| = O(n^\alpha)$ and $|Y| = O(n^\beta)$, such a dynamic matrix problem is called α, β-*restricted*.

Theorem 4. *The α, β-restricted problems of dynamic matrix inverse and matrix adjoint over a field, with non-singular simple updates, can be solved with the following costs:*

- **initialization** $O(n^\omega)$ *arithmetic operations,*
- **update** $O(n^{\omega(1,\varepsilon,1)-\varepsilon} + n^{1+\varepsilon} + n^{\alpha+\beta})$ *arithmetic operations (worst-case),*
- **query for inverse and adjoint** $O(1)$ *arithmetic operations (worst-case).*

Proof. We use the algorithm from Theorem 2 to maintain the inverse matrix. We will now show how to recompute $(A^{-1})_{X,Y}$ after each update with the use of this algorithm. The matrix A'^{-1} is given by

$$A'^{-1} = B^{-1}A^{-1} = (I + B^{-1} - I)A^{-1} = A^{-1} + (B^{-1} - I)A^{-1}.$$

Notice that due to a special form of B^{-1} (B has non-zero elements only on diagonal and i-th column) only the i-th row of A^{-1} is used in this multiplication. In other words, we have

$$A'^{-1} = A^{-1} + ((B^{-1} - I))_i (A^{-1})^i.$$

This allows to recompute only a part of A^{-1} in the following way

$$(A'^{-1})_{X,Y} = (A^{-1})_{X,Y} + \left((B^{-1} - I)\right)_{X,\{i\}} A^{-1}_{\{i\},Y},$$

Querying out $(A^{-1})_{\{i\},Y}$ takes $O(n^{1+\epsilon})$ arithmetic operations and does not increase update cost (see Theorem 2). The recomputation of the part of the inverse requires only a vector-vector multiplication which costs $O(|X||Y|) = O(n^{\alpha+\beta})$ arithmetic operations.

4 Dynamic Shortest Paths

We are now ready to introduce a dynamic algorithm for computing the shortest path lengths in unweighted graphs. A *symbolic adjacency matrix* of the directed graph G is the $n \times n$ matrix $\tilde{A}$ such that

$$\tilde{A}_{i,j} = \begin{cases} x_{i,j} \text{ if } (v_i, v_j) \in E, \\ 0 \text{ otherwise,} \end{cases}$$

where $x_{i,j}$ are unique variables corresponding to the edges of G.

Theorem 5. *Let $\tilde{A}$ be the symbolic adjacency matrix of a graph G. Consider* $\mathrm{adj}(I-\tilde{A}u)_{i,j}$ *as the polynomial of u. The length of the shortest path in G from i to j is equal to the degree of the smallest degree term in* $\mathrm{adj}(I-\tilde{A}u)_{i,j}$. *Moreover, all non-zero terms in* $\mathrm{adj}(I-\tilde{A}u)_{i,j}$ *are also non-zero over a finite field $\mathcal{Z}_p$.*

Proof. We have $\mathrm{adj}(I-u\tilde{A})_{i,j} = (-1)^{i+j}\det((I-u\tilde{A})^{j,i})$. Equivalently, if we take $\tilde{Z}$ to be the matrix obtained from $I-u\tilde{A}$ by zeroing entries of the j'th row and the i'th column and setting the entry (j,i) to one, then $\mathrm{adj}(I-u\tilde{A})_{i,j} = \det(\tilde{Z})$, and

$$\mathrm{adj}(I-u\tilde{A})_{i,j} = \sum_{p\in\Gamma_n} \mathrm{sgn}(p)\prod_{k=1}^{n} z_{k,p_k}. \tag{7}$$

Take a permutation p such that $\prod_{k=1}^{n} z_{k,p_k} \neq 0$. The permutation p can be viewed as a set of cycles. Notice that the above product is non-zero if $p_j = i$ for some j, so there exists a cycle c containing (j,i) in p. The rest of c forms a path from i to j. There may exist many permutations containing c that give a non-zero contribution to the sum. However, the smallest degree term is introduced by the permutation that is identity on vertices v not belonging to c, i.e., $p_v = v$. The degree of this term is the length of the path from i to j in c. Hence the smallest degree term in the sum corresponds to the shortest path.

Notice that each monomial in (7) has coefficient 1, so each non-zero term in $\mathrm{adj}(I-u\tilde{A})_{i,j}$ is also non-zero over $\mathcal{Z}_p$.

The above theorem gives us the connection between matrix adjoint and shortest path lengths. However, in order to construct an efficient algorithms we cannot do computations symbolically. The standard way of solving this problem is to use following lemma due to Zippel [19] and Schwartz [16].

Lemma 2. *If $p(x_1,\ldots,x_m)$ is a non-zero polynomial of degree d with coefficients in a field and S is a subset of the field, then the probability that p evaluates to 0 on a random element $(s_1,s_2,\ldots,s_m) \in S^m$ is at most $d/|S|$. We call such event a false zero.*

It follows from the lemma that if we evaluate a polynomial of degree n over random variables modulo prime number p of length $(1+c)\log n$ we get a false zero with probability at most $\frac{1}{n^c}$, for any $c > 0$.

In order to compute distances in G dynamically, we can proceed as follows. We generate a random adjacency matrix A from the symbolic adjacency matrix $\tilde{A}$ of G by substituting each nonzero entry in $\tilde{A}$ with a random number in the range $1,\ldots,p-1$. Note that the matrix $I-uA$ has the same form as the matrix maintained in Theorem 1. Thus we can use the same algorithm in order to maintain dynamically the adjoint of the matrix $I-uA$. The queries for the distances in G can be answered by finding the smallest degree term in $\mathrm{adj}(I-uA)_{i,j}$. Similarly as in Theorem 1 we do the computation over $R^n[[u]]$. However, if the computations are done over $R^k[[u]]$, for any $k \leq n$, then the algorithm computes the shortest distances up to the length k. This truncation is correct because the terms of the given degree depend only on the terms of degree less or equal. Thus we get the following theorem.

Theorem 6. *There exists an algorithm for maintaining dynamic shortest distances $\leq k$ in unweighted graphs with updates in $\tilde{O}(k(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon}))$ time and queries in $\tilde{O}(kn^{\epsilon})$ time. The algorithm is randomized and with small probability may return wrong, larger distances.*

Proof. To obtain the above bound we use the lazy computation schemes from Theorem 3. The update time and query time follow from the fact that the arithmetic operations over the formal power series modulo u^{k+1} can be carried out in time $\tilde{O}(k)$. In the queries we have to compute the degree of the smallest degree term in an element from $R^k[[u]]$ and this takes $O(k)$ time.

Remark 1. Theorem 6 can be also used to maintain the part of the distance matrix, similarly as it was stated in Theorem 4. The update cost for α, β-restricted version of this problem is $\tilde{O}(k(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon} + n^{\alpha+\beta}))$ whereas the query cost is $\tilde{O}(k)$.

5 Sub-quadratic Algorithm for Shortest Lengths

In our algorithm we use the decomposition technique introduced in [8], and later used in e.g., [4, 5, 9, 11, 18, 20].

Theorem 7 (Ullman and Yannakakis [18]). *Let $H \subseteq V$ be a set of vertices chosen uniformly at random. Then the probability that a given simple path has a sequence of more than $\frac{cn}{|H|} \log n$ vertices, none of which is from H, for any $c > 0$, is , for sufficiently large n, bounded by $2^{-\alpha c}$ for some positive α.*

We are now ready to prove the following theorem.

Theorem 8. *There exists an randomized algorithm for the dynamic shortest distances problem in unweighted graph $G = (V, E)$ supporting edge updates in $O(n^{1.932})$ and queries in $O(n^{1.288})$ time.*

Proof. The algorithm is based on the idea used in construction of the $\mathcal{F}_2$ family of algorithms in [5]. In the algorithm we maintain:

- A set $H \subseteq V$ of vertices chosen uniformly at random with $|H| = \frac{cn}{n^{\mu}} \log n = \tilde{O}(n^{1-\mu})$, for any constant $c > 0$, where $c \log n \leq n^{\mu} \leq n$.
- An $n \times n$ matrix D such that $D_{i,j}$ is the length of the shortest path from i to j in G, that uses at most n^{μ} edges.
- An $|H| \times |H|$ matrix B obtained from D by choosing only columns and rows corresponding to H.
- The Kleene closure B^* of matrix B ,i.e. the distance matrix obtained from B.

In each update:

- We update the matrix D using the algorithm from Theorem 6. This takes $\tilde{O}(n^{\mu}(n^{1+\epsilon} + n^{\omega(1,\epsilon,1)-\epsilon}))$ time.

- The cost of maintaining the matrix B is $\tilde{O}(n^{\mu}n^{2-2\mu})$ as stated in Remark 1 and Theorem 4.
- We recompute the matrix B^* from scratch in $O(n^{3-3\mu})$ time.

The query on the distance from i to j can be answered by computing

$$\min\left\{D_{i,j}, \min_{p,q\in H}\left\{D_{i,p} + B^*_{p,q} + D_{q,j}\right\}\right\},$$

where we have to query out a row and a column from D, and then compute the minimum over p, q, what takes $O(n^{1-\mu+\epsilon} + n^{2-2\mu})$ operations.

We get the following bound on time for the edge update:

$$\tilde{O}(n^{\mu}(n^{1+\epsilon} + n^{\omega(1,\epsilon,1)-\epsilon}) + n^{2-2\mu}n^{\mu} + O(n^{3-3\mu})).$$

In order to get the fastest update time, we take $\epsilon = 0.575$. This balances the first two terms. By balancing the first and the last term we get

$$1 + \epsilon + \mu = 3 - 3\mu,$$

$$4\mu = 2 - \epsilon,$$

$$\mu = 0.357.$$

The first term is now equal to $O(n^{1.932})$, the third term is smaller and equal to $O(n^{1.644})$, whereas the query time is $O(n^{1.288})$.

6 Summary and Conclusions

We have presented for the first time dynamic algorithms for computing: matrix determinant and matrix adjoint over commutative rings. The algorithm can be extended to work over Euclidean rings.

Using these algorithms we were able to solve dynamic shortest path problem in unweighted graphs in $o(n^2)$ time per update and per query, thus solving a long standing open question if such algorithms exist. However, we have only solved the easy case of unweighted graphs. Notice that the algorithm can be easily modified for the case of small integer weights, but the question of existence of such algorithms for graphs with arbitrary weights remains open. It is also not known if the problem of single source shortest distances can be solved more efficiently, e.g., update in $o(n^2)$ time and query in constant time.

Acknowledgments

I would like to thank my favorite supervisor Krzysztof Diks for his unwavering support and Marcin Mucha for many helpful discussions. I would also like to thank Uri Zwick for finding an error in the first version of this paper.

References

1. J. Bunch and J. Hopcroft. Triangular Factorization and Inversion by Fast Matrix Multiplication. *Math. Comp.*, 28:231–236, 1974.
2. D. Coppersmith. Rectangular Matrix Multiplication Revisited. *J. Complex.*, 13(1):42–49, 1997.
3. D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of Computing*, pages 1–6. ACM Press, 1987.
4. C. Demetrescu and G.F. Italiano. Fully Dynamic All Pairs Shortest Paths with Real Edge Weights. In *Proceedings of 42th annual IEEE Symposium on Foundations of Computer Science*, pages 260–267, 2001.
5. C. Demetrescu and G.F. Italiano. Improved Bounds and New Trade-Offs for Dynamic All Pairs Shortest Paths. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 633–643. Springer-Verlag, 2002.
6. C. Demetrescu and G.F. Italiano. A new Approach to Dynamic all Pairs Shortest Paths. In *Proceedings of the thirty-fifth annual ACM Symposium on Theory of Computing*, pages 159–166. ACM Press, 2003.
7. G.S. Frandsen, J.P. Hansen, and P.B. Miltersen. Lower Bounds for Dynamic Algebraic Problems. *Lecture Notes in Computer Science*, 1563:362–372, 1999.
8. D.H. Greene and D.E. Knuth. *Mathematics for the Analysis of Algorithms.* Birkhäuser, 1982.
9. M.R. Henzinger and V. King. Fully Dynamic Biconnectivity and Transitive Closure. In *Proceedings 36th annual IEEE Symposiumon Foundations of Computer Science*, pages 664–672, 1995.
10. X. Huang and V.Y. Pan. Fast Rectangular Matrix Multiplication and Applications. *Journal of complexity*, 14(2):257–299, 1998.
11. V. King. Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs. In *Proceedings of 40th annual IEEE Symposium on Foundations of Computer Science*, pages 81–91, 1999.
12. H.T. Kung and J.F. Traub. All Algebraic Functions Can Be Computed Fast. *J. ACM*, 25(2):245–260, 1978.
13. J.H. Reif and S.R. Tate. On Dynamic Algorithms for Algebraic Problems. *J. Algorithms*, 22(2):347–371, 1997.
14. P. Sankowski. Dynamic Transitive Closure via Dynamic Matrix Inverse. In *Proceedings of the 45th annual IEEE Symposium on Foundations of Computer Science*, pages 509–517, 2004.
15. A. Schonhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.
16. J.T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. Algorithms*, 10:701–717, 1980.
17. V. Strassen. Vermeidung von Divisionen. *J. reine u. angew. Math.*, 264:182–202, 1973.
18. J. Ullman and M. Yannakakis. High-probability Parallel Transitive Closure Algorithms. In *Proceedings of the 1990 ACM Symposium on Parallel Algorithms and Architectures*, pages 200–209, July 1990.
19. R.E. Zippel. Probabilistic Algorithms for Sparse Polynomials. *Proc. EUROSAM 79, Lecture Notes in Computer Science 72*, pages 216–226, 1979.
20. U. Zwick. All Pairs Shortest Paths in Weighted Directed Graphs Exact and Almost Exact Algorithms. In *Proceedings of the 39th annual IEEE Symposium on Foundations of Computer Science*, pages 310–319, 1998.

Randomly Generating Triangulations of a Simple Polygon

Q. Ding[1], J. Qian[2], W. Tsang[3], and C. Wang[2]

[1] Zhejiang Radio & TV Transmission Center, HangZhou, China
dingqinh@163.net
[2] Department of Computer Science, Memorial University of Newfoundland
St. John's, Newfoundland, Canada A1B 3X5
{jianbo,wang}@garfield.cs.mun.ca
[3] Department of Computer Science, The University of Hong Kong
Hong Kong, China
tsang@cs.hku.hk

Abstract. In this paper, we present an $O(n^2+|E|^{\frac{3}{2}})$ time algorithm for generating triangulations of a simple polygon at random with uniform distribution, where n and $|E|$ are the number of vertices and diagonal edges in the given polygon, respectively. The current best algorithm takes $O(n^4)$ time. We also derive algorithms for computing the expected degree of each vertex, the expected number of ears, the expected number of interior triangles, and the expected height of the corresponding tree in such a triangulated polygon. These results are not known for simple polygon. All these algorithms are dominated by the $O(n^2 + |E|^{\frac{3}{2}})$ time triangulation counting algorithm. If the results of the triangulation counting algorithm are given, then the triangulation generating algorithm takes $O(n \log n)$ time only. All these algorithms are simple and easy to be implemented.

1 Introduction

The triangulation of a simple polygon is a fundamental structure in computational geometry. The efficiency of many important algorithms is based on triangulated structures. These algorithms are point location, ray-shooting, and visibility area computing in a simple polygon, just naming a few. While theoretical analysis of these algorithms indicated they are correct and efficient, their implementations may need special care. Some types of input triangulations may significantly slow down these algorithms or may even cause errors. One needs to know what these cases are and how often they occur. Randomly-generated triangulations with uniform distribution are a good choice for such unbiased tests.

While generating triangulations of a convex polygon at random with uniform distribution can be done in linear time [DFHNS, Ep] the best algorithm for generating triangulations of a simple polygon takes $O(n^4)$ time, proposed by Epstein and Sack [ES].

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 471–480, 2005.

In order to find out the average performance of some algorithms based on triangulations of a simple polygon, researchers also need to know the expected values of some parameters of these triangulations such as the expected degree of a vertex and the expected height of the corresponding tree. Devroye [De, DFHNS] investigated these parameters for convex polygons, but no result is known to the author's best knowledge for simple polygons.

In Section 2.1 of this paper, we present an $O(n^2 + |E|^{\frac{3}{2}})$ time algorithm for counting the number of triangulations in a given simple polygon as well as its subpolygons. If the visibility graph is given, then the counting algorithm takes $O(|E|^{\frac{3}{2}})$ time. In Section 2.2, we propose an $O(n \log n)$ algorithm for generating triangulations of a simple polygon at random with uniform distribution. In Sections 2.3 to 2.6, we demonstrate algorithms for computing the expected values of the following parameters: the degree of a vertex (which takes $O(|E|)$ time), and the number of ears, the number of internal triangles, and the height of the corresponding tree (which all take $O(|E|^{\frac{3}{2}})$ time). Section 3 gives our concluding remarks.

2 Algorithms

We shall first give some definitions and notations.

Let $P = (p_1, p_2, ...p_n)$ be the vertices of a simple polygon P in clockwise order. We shall assume for the rest of this paper that $p_{i+kn} = p_i$ for integer k, that is, the vertices of the polygon are enumerated modulo n.

A *diagonal* is a line segment connecting two vertices of P and lying inside P. The boundary edge set of P is denoted by B.

Let $G = (V, E)$ be the visibility graph of P, where V is the vertex set of P and E is the diagonal set and boundary edge set of P.

Let $P(i, j)$ stand for the subpolygon of P induced by the chain of vertices $p_i, p_{i+1}, .., p_j$ for $1 \leq i, j \leq n$ and $p_i p_j \in E$. Let $t(i, j)$ be the number of triangulations of $P(i, j)$.

Let $T(P)$ be a triangulation of P, and $t_e \in T(P)$ be a triangle. We call t_e an *ear* if two edges of t_e are boundary edges of P, and an *internal triangle* if all of its three edges are diagonals. The degree in $T(P)$ of a vertex p is the number of diagonals in $T(P)$ incident to p.

Note that if the total number of triangulations and the number of triangulations containing a specific triangle is known, the probability of this triangle appearing on a triangulation generated at random with uniform distribution can be easily calculated. If we know this probability for each triangle, we can randomly generate triangulations with uniform probability. Thus, we shall first present an algorithm for counting the number of triangulations. Also note that the size of edge set of the visibility graph determines the number of triangles and triangulations [BE]. Therefore, the expected values of the above mentioned parameters depends on the edge set of the visibility graph.

2.1 Counting the Number of Triangulations in Polygon P

Given a simple polygon P with n vertices, we shall first preprocess P in $O(n^2)$ time to obtain the following outputs: (1) the visibility graph of P, $G(V, E)$, (2)

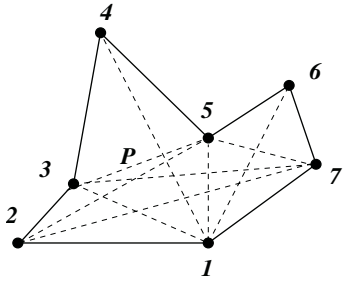
a sorted list E', consisting of p_ip_j and p_jp_i for each pair of vertices p_i and p_j visible to each other, according to the difference of the two vertex indexes of an edge in ascending order, and (3) $|E'|$ sets of vertices, in which vertex set $V(i,j)$ is attached to edge $p_ip_j \in E$ such that each vertex p_k in $V(i,j)$ together with edge p_ip_j form a triangle $p_ip_kp_j$ in $P(i,j)$. (Note that $|E'| = 2|E|$ since p_ip_j and p_jp_i are different in E'. Edge p_ip_j is attached by polygon $P(i,j)$ and vertex set $V(i,j)$, and edge p_jp_i is attached by polygon $P(j,i)$ and vertex set $V(j,i)$. For instance in Figure 1, $E' = (p_1p_2, ..., p_7p_1, p_1p_3, p_3p_5, ..., p_7p_2, p_1p_4, ..., p_7p_3, ..., p_2p_1, ..., p_1p_7)$.))

Pre-Process-Algorithm(P)
Method:
1 Find the visibility graph of P, $G(V,E)$.
2 Find the sorted list E'.
3 Find $V(i,j)$ for each p_ip_j in E'.
(* $deg_G(p_i)$ denotes the degree of vertex p_i in $G(V,E)$ *)
For $i = 1$ **To** n **Do**
  If $deg_G(p_i) \geq \sqrt{|E|}$ **Then**
    For each $p_jp_k \in E'$ not incident to p_i **Do**
      If $p_jp_i, p_kp_i \in E'$ **And** $p_ip_jp_k$ in clockwise order **Then** Add p_j to $V(i,k)$
    End For
  Else
    For each pair $p_ip_j, p_ip_k \in E'$ incident to p_i **Do**
      If $p_jp_k \in E'$ **And** $p_ip_jp_k$ in clockwise order **Then** Add p_j to $V(i,k)$
    End For
  End If
End For

Pre-Process-Algorithm(P) takes $O(n^2)$ time. Step 1 can be done in $O(n^2)$ time [We]. Step 2 takes $O(n^2)$ time if the output of visibility graph is an adjacency matrix. Step 3 takes $O(|E|^{\frac{3}{2}})$ time since in the **If** block, there are at most $|E|$ edges not incident to vertex p_i and there are at most $O(\sqrt{|E|})$ such vertices as p_i, and in the **Else** block, for each edge p_ip_k together with another edge incident to p_i can form at most $\sqrt{|E|}$ triangles since $deg_G(p_i) < \sqrt{|E|}$, and there are at most $|E|$ such edges as p_ip_k. (Remark: the proof of the upper bound $O(|E|^{\frac{3}{2}})$ for the number of triangles in P can be found in [BE].)

With the output of **Pre-Process-Algorithm**(P), we design the following efficient dynamic programming algorithm to calculate $t(i,j)$ for all $p_ip_j \in E'$. Furthermore, with a constant factor of extra cost, the algorithm can produce $|E'|$ sequences of probabilities, such that the probability $prob_k(i,j)$ in $Prob(i,j)$ is attached to triangle $p_ip_kp_j$ in $P(i,j)$. Let $prob_k(i,j)$ be the probability of triangle $p_ip_kp_j$ appearing in a random triangulation $T(P(i,j))$ with uniform probability. For every $p_ip_j \in E'$, we also produce a sequence $(C_l(i,j))$ of cumulative sum of $prob_k(i,j)$.

Algorithm Tri-Count(P)
Input: $E', \{V(i,j)\}$
Output: $t(i,j)$, $\{Prob(i,j)\}$, and $(C_l(i,j))$ attached to edge p_ip_j for all $p_ip_j \in E'$.
Method:
While $E' \neq \emptyset$ **Do**
  $e \leftarrow$ **extract**(E'); Let $e = p_ip_j$
  If $j = i+1$ **Or** $j = i+2$ **Then**
  $t(i,j) = 1$
  Else
    $t(i,j) = 0$; $V'(i,j) \leftarrow V(i,j)$
    While $V(i,j) \neq \emptyset$ **Do**
      $p_k \leftarrow$ **extract**$(V(i,j))$
      $t(i,j) \leftarrow t(i,j) + t(i,k)t(k,j)$
    End While
    $l \leftarrow 0$; $c_l(i,j) \leftarrow 0$
    While $V'(i,j) \neq \emptyset$ **Do**
      $p_k \leftarrow$ **extract**$(V'(i,j))$
      $Prob_k(i,j) \leftarrow t(i,k)t(k,j)/t(i,j)$
      $l \leftarrow l+1$
      $c_l(i,j) \leftarrow c_{l-1}(i,j) + prob_k(i,j)$
    End While
  End If
End While



j \ i	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	0	1	0	1
3	1	1	1	1	1	0	1
4	1	0	1	1	1	0	0
5	1	1	1	1	1	0	1
6	1	0	0	0	1	1	1
7	1	1	1	0	1	1	1

The Visibility graph, 0–invisible, 1–visible.

j \ i	1	2	3	4	5	6	7
1	0	9	5	2	2	1	1
2	1	0	9	0	3	0	1
3	1	1	0	9	7	0	2
4	1	0	1	0	9	0	0
5	3	1	1	1	0	9	6
6	3	0	0	0	1	0	9
7	9	2	1	0	1	1	0

The t(i,j) table, where the numbers are the triangulations.

Fig. 1. An illustration of $t(i,j)$ values of a simple polygon.

Lemma 1. *Algorithm Tri-Count(P) takes* $O(|E|^{\frac{3}{2}})$ *time to calculate the values* $t(i,j)$ *for all* $p_ip_j \in E$ *of a simple polygon* P *with* n *vertices and* $|E|$ *diagonals.*

Proof. For the time complexity, the outer-while-loop iterates $|E'|$ times and inner-while-loops are bounded by $|V(i,j)|$. Thus, the total time is $\sum_{p_ip_j \in E'} |V(i,j)| = c \cdot |E'|^{\frac{3}{2}}$ time for some constant c, *i.e.*, the number of trian-

gles in P [BE]. Thus **Tri-Count(P)** takes $O(|E|^{\frac{3}{2}})$ time. The correctness is due to the recurrence $t(i,j) = \sum_k t(i,k) \cdot t(k,j)$. □

2.2 Randomly Generating Triangulations of P with Uniform Probability

It is not difficult to see that the probability that a triangle $p_ip_jp_k$ appears in a triangulation of $P(i,j)$ generated randomly with uniform distribution is the ratio of the number of triangulations for subpolygon $P(i,k)$ multiplied by the number of triangulations for subpolygon $P(k,j)$ over the number of triangulations for polygon $P(i,j)$.

Lemma 2. *Suppose $p_ip_j \in E$, $p_ip_k \in E$ and $p_kp_j \in E$, then triangle $p_ip_jp_k$ appears in a uniformly random triangulation of $P(i,j)$ at the probability of:*

$$prob_k = \frac{t(i,k)t(k,j)}{t(i,j)}$$

Suppose the indexes of vertices in $V(i,j)$ are $(k_1, k_2, ..., k_{|V(i,j)|})$ by the order of appearance. In order to form a tria ngulation $T(P(i,j))$, at least one triangle, say $p_ip_kp_j$ for some k, must exist. Then the summation $\sum_k prob_k$ of the probabilities in $Prob(i,j)$ is 1. We can treat $Prob(i,j)$ as a unit interval divided into $|V(i,j)|$ intervals $(I_{k_1}, I_{k_2}, ..., I_{k_{|V(i,j)|}})$, where the length of I_{k_m} is $prob_{k_m}$. For example in Figure 1, if $V(2,1) = (p_3, p_7, p_5)$, then $Prob(2,1)$ is divided into $I_3 = [0, 5/9]$, $I_7 = [5/9, 7/9]$ and $I_5 = [7/9, 1]$, where $5/9, 7/9$ and 1 are exactly the cumulative sums $c_1 = 5/9$, $c_2 = 5/9+2/9$ and $c_3 = 5/9+2/9+2/9$ produced in Tri-Count. The length of an interval I_k in $Prob(i,j)$ represents the probability $prob_k$ that I_k being randomly chosen in interval $[0,1]$ with uniform probability. Let u be a random number generated in interval [0,1] uniformly. If u matches the interval I_k, then it implies that p_k is randomly picked with uniform probability.

By Lemma 2 and if we know $\{Prob(i,j)\}$, we can recursively triangulate $P(i,j)$ at random with uniform probability. The algorithm triangulates P when we set $j = i - 1$.

Algorithm Rand-Tri($P(i,j)$)
Input: simple polygon $P(i,j)$, E', and $Prob(k,m)$ for all $p_k, p_m \in P(i,j)$.
Output: a random triangulation of $P(i,j)$ at uniform probability.
Method:
If $j = i+1$ **Or** $j = i+2$ **Then Return** (* boundary conditions*)
Else
    $u \leftarrow uniform[0,1]$;
    Binary search of u on $\{I_l\}$; Suppose $u \in I_k$. Insert p_ip_k and p_kp_j as diagonals
    (* triangle $p_ip_kp_j$ is chosen at the probability of $prob_k$*)
    Rand-Tri($P(i,k)$) (* recursion *)
    Rand-Tri($P(k,j)$)
End If

Lemma 3. *Algorithm Rand-Tri*$(P(i,j))$ *randomly triangulates a simple polygon* $P(i,j)$ *at uniform probability in* $O(|P(i,j)| \log max\{|V(i,j)|\})$ *time.*

Proof. For the time complexity, each recurrence produces a triangle which separated $P(i,j)$ into two subpolygons. There are $j-i-1$ triangles in any triangulation. The binary search of u on $Prob(i,j)$ takes logarithmic time. The correctness is due to Lemma 2. □

2.3 Calculating the Expected Degrees of Vertices in $T(P)$

Let $d(i)$ denote the expected degree of vertex p_i in P, that is, the average diagonals incident to p_i for all triangulations of P. Obviously, $\sum_{p_i \in P} d(i) = 2 \cdot (n-3)$.

The following lemma gives the equation to calculate $d(i)$ in P. That is, the probability that an edge p_ip_k appeared in all triangulations in P is the ratio of the number of triangulations in $P(i,k)$ multiplied by the number of triangulations in $P(k,i)$ divided by the number of triangulations in $P = (P(i,i-1))$.

Lemma 4. *Let* $K = \{k \mid i+2 \le k \le i-2; p_ip_k \in E'\}$. *We have that*

$$d(i) = \sum_{k \in K} \frac{t(i,k)t(k,i)}{t(i,i-1)}.$$

By Lemma 4, we can calculate $d(i)$ using the output of Algorithm Tri-Count. This calculation takes $O(|E|)$ time since only diagonal $p_ip_k \in E$ needs to be calculated and there are $|E|$ such diagonals.

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*
d(i)	$\frac{16}{9}$	$\frac{5}{9}$	$\frac{14}{9}$	$\frac{2}{9}$	$\frac{22}{9}$	$\frac{3}{9}$	$\frac{10}{9}$

Fig. 2. The expected degrees of vertices in the simple polygon in Figure 1.

2.4 Calculating the Expected Number of Ears in $T(P)$

The expected number of ears in P is defined as the ratio of the number of ears in all triangulations over the number of all triangulations in P.

To calculate the expected number of ears recursively, we consider a subpolygon $P(i,j)$. Let $er(i,j)$ denote the expected number of ears whose edges do not include p_ip_j. To recursively calculate the expected ears, consider triangle $p_ip_kp_j$ appears in some triangulations. $p_ip_kp_j$ divides $P(i,j)$ into $P(i,k)$ with expected ears $er(i,k)$ and $P(k,j)$ with expected ears $er(k,j)$. The probability of the above case appearing in all triangulations of $P(i,j)$ is $(t(i,k)t(k,j)/t(i,j))$. Therefore,

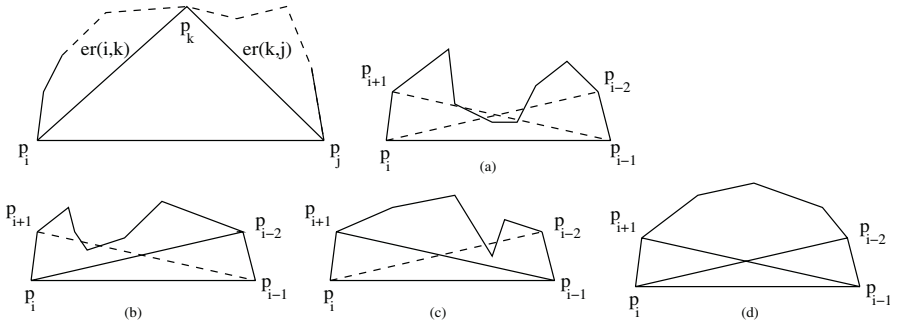


Fig. 3. The recurrence for finding the expected ears.

we have recurrence (1) in the lemma. Furthermore, let $er'(i, i-1)$ denote the expected number of ears whose edges include p_ip_{i-1} (a boundary edge), there are four cases (Figure 3) to calculate $er'(i, i-1)$. Each case has its probability in the triangulations of $P(i, i-1)$ as shown in the part (2) of the following lemma. The expected number of ears of polygon P) is $er(i, i-1) + er'(i, i-1)$.

Lemma 5. *Suppose* $p_ip_j \in E'$, *let* $K = \{k \mid p_k \in V(i,j)\}$. *Then,*

$$(1) \quad er(i,j) = \sum_{k \in K} \frac{t(i,k)t(k,j)}{t(i,j)}(er(i,k) + er(k,j)).$$

$$(2) \begin{cases} (a) er'(i,i-1) = 0 & (p_ip_{i-2} \notin E)\& \\ & (p_{i+1}p_{i-1} \notin E); \\ (b) er'(i,i-1) = t(i,i-2)/t(i,i-1) & (p_ip_{i-2} \in E)\& \\ & (p_{i+1}p_{i-1} \notin E); \\ (c) er'(i,i-1) = t(i+1,i-1)/t(i,i-1) & (p_ip_{i-2} \notin E)\& \\ & (p_{i+1}p_{i-1} \in E); \\ (d) er'(i,i-1) = t(i+1,i-1)/t(i,j)+t(i,i-2)/t(i,i-1) & (p_ip_{i-2} \in E)\& \\ & (p_{i+1}p_{i-1} \in E). \end{cases}$$

By Lemma 5, we can calculate $er(i,j)$ by dynamic programming, using the output of Algorithm Tri-Count. This algorithm takes $O(|E|^{\frac{3}{2}})$ time.

Algorithm Calc-Expt-Ears.
Input: $t(i,j)$, $\{V(i,j)\}$ for all $p_ip_j \in E'$.
Output: $er(i,j)$ for all $p_ip_j \in E'$.
Method:
While $E' \neq \emptyset$ **Do**
  $e \leftarrow$ **extract**(E') {* Let $e = p_ip_j$ *}
  $er(i,j) \leftarrow 0$
  While $V(i,j) \neq \emptyset$ **Do**
    $p_k \leftarrow$ **extract**$(V(i,j))$
    $er(i,j) \leftarrow er(i,j) + \frac{t(i,k)t(k,j)}{t(i,j)}(er(i,k) + er(k,j))$
  End While
End While

2.5 Finding the Expected Number of Internal Triangles in $T(P)$

We can define the expected number of internal triangles similar to that of the expected number of ears in subpolygon $P(i,j)$. Let $i_n(i,j)$ denote the expected number of internal triangles in subpolygon $P(i,j)$, that is, the ratio of the number of internal triangles in all triangulations over the number of triangulations of $P(i,j)$. Let $i'_n(i,j)$ denote the expected number of triangles with p_ip_j and two other diagonals as edges in subpolygon $P(i,j)$. Note that such a triangle is an internal triangle of P if p_ip_j is a diagonal in P and it is not an internal triangle if p_ip_j is a boundary edge.

The following lemma gives the recursive equations to calculate $i_n(i,j)$ and $i'_n(i,j)$:

Lemma 6. *Suppose $p_ip_j \in E'$, let $K = \{k \mid p_k \in V(i,j)\}$, and let $K' = \{k \mid i+2 \le k \le j-2; p_k \in V(i,j)\}$. We have that*

$$\begin{cases} i_n(i,j) = \sum_{k\in K} \frac{t(i,k)t(k,j)}{t(i,j)}(i_n(i,k) + i_n(k,j) + i'_n(i,k) + i'_n(k,j)). \\ i'_n(i,j) = \sum_{k\in K'} \frac{t(i,k)t(k,j)}{t(i,j)}. \end{cases}$$

By Lemma 6, we can calculate all $i_n(i,j)$ and $i'_n(i,j)$ for $p_ip_j \in E'$ by dynamic programming, using the output of Algorithm Tri-Count (note that $i_n(i,i-1)$ is exactly the expected number of internal triangles in P, for any $1 \le i \le n$.):

Algorithm Calc-Expt-Int-Tri.
Input: $t(i,j)$, $\{V(i,j)\}$ for all $p_ip_j \in E'$.
Output: $i_n(i,j)$ and $i'_n(i,j)$ for all $p_ip_j \in E'$.
Method:
While $E' \neq \emptyset$ **Do**
  $e \leftarrow$ **extract**(E'); Let $e = p_ip_j$
  $i_n(i,j) \leftarrow 0$; $i'_n(i,j) \leftarrow 0$
  While $V(i,j) \neq \emptyset$ **Do**
    $p_k \leftarrow$ **extract**$(V(i,j))$
    $i_n(i,j) \leftarrow i_n(i,j) + \frac{t(i,k)t(k,j)}{t(i,j)}(i_n(i,k) + i_n(k,j) + i'_n(i,k)$
    $+i'_n(k,j))$
    If $p_k \neq p_{i+1}$ **And** $p_k \neq p_{j-1}$ **Then** $i'_n(i,j) \leftarrow i'_n(i,j) + \frac{t(i,k)t(k,j)}{t(i,j)}$
  End While
End While

2.6 Calculating the Expected Height of the Rooted Tree of $T(P)$

For a triangulation $T(P(i,j))$, let $t_T \in T(P(i,j))$ be the triangle containing p_ip_j as one of its edges. Let R_T be the dual graph (rooted tree) of T that has t_T as its root, and let $h_T(i,j)$ be its height. Let $h(i,j)$ denote the expected height of R_T for a random triangulation $T(P(i,j))$ with uniform probability, that is, the average value of $h_T(i,j)$ for all triangulations of $P(i,j)$.

The following lemma gives the recursive equation to calculate $h(i,j)$ (for convenience, we assume that $h(i,i+1) = -1$).

Lemma 7. *Suppose* $p_ip_j \in E'$, *let* $K = \{k \mid p_k \in V(i,j)\}$. *We have that*

$$h(i,j) = \sum_{k \in K} \frac{t(i,k)t(k,j)}{t(i,j)}(Max\{h(i,k), h(k,j)\} + 1).$$

By Lemma 7, we can calculate $h(i,j)$ by dynamic programming, using the output of Algorithm Tri-Count.

Algorithm Calc-Expt-Height.
Input: $t(i,j)$, $\{V(i,j)\}$ for all $p_ip_j \in E'$.
Output: $h(i,j)$ for all $p_ip_j \in E'$.
Method:
While $E' \neq \emptyset$ **Do**
  $e \leftarrow$ **extract**(E'); Let $e = p_ip_j$
  If $j = i + 1$ **Then**
    $h(i,j) = -1$
  Else
    $h(i,j) \leftarrow 0$
    While $V(i,j) \neq \emptyset$ **Do**
      $p_k \leftarrow$ **extract**$(V(i,j))$
      $h(i,j) \leftarrow h(i,j) + \frac{t(i,k)t(k,j)}{t(i,j)}(Max\{h(i,k), h(k,j)\} + 1)$
    End While
  End If
End While

3 Conclusion Remark

In this paper, we present an $O(|E|^{\frac{3}{2}})$ dynamic programming algorithm for counting the number of triangulations of a simple polygon, given its visibility graph. This algorithm improves on the $O(n^3)$ existing algorithm [ES]. Based on this algorithm, we devise an $O(n \log n)$ random triangulation generation algorithm with uniform distribution, which improves on the existing $O(n^4)$ algorithm. We also proposed algorithms for computing the expected values of various parameters of triangulations of simple polygons. All these algorithms use dynamic programming.

The idea of our **Pre-Process-Algorithm** can be used to improve the optimal triangulation algorithm proposed by by Bern and Eppstein [BE] from $O(n^2 + |E|^{\frac{3}{2}})$ time to $O(|E|^{\frac{3}{2}})$, given the visibility graph (Theorem 5 in [BE]). We can use a similar method to randomly generate triangulations of a simple polygon with binary-tree-search distribution, and for finding the MaxMin / MinMax of some parameters.

The open problem is how to extend this method to 3D case. Even for 3D convex polyhedra, the problem is challenging.

Acknowledgement

This work is partially supported by NSERC discovery grant OPG0041629 and HK RGC grant HKU 7143/04E.

References

1. Atkinson M. and Sack J., 'Generating Binary Trees at Random', *Information Processing Letters*, 41 (1992), pp.21-23.
2. Devroye L., 'A Note on the Height of Binary Search Trees', *Journal on ACM*, 33 (1986), pp. 489-498.
3. Devroye L., Flajolet P., Hurtado F., Noy M., and Steiger W., 'Random Triangulations', *Discrete and Computational Geometry*, Vol. 22, pp.105-117, 1999.
4. Epstein P. and Sack J., 'Generating triangulation at Random', *ACM Trans. On Modeling and Computer Simulation*, Vol.4 (1994), No.3, pp.267-278.
5. Epstein P., 'Generating geometric objects at random', Master's thesis (1992),School of Computer Science, Carleton university, Canada.
6. Bern M. and Eppstein D., 'Mesh Generation And Optimal Triangulation', Computing in Euclidean Geometry, Edited by Ding-Zhu Du and Frank Hwang, World Scientific, Lecture Notes Series on Computing – Vol.1.
7. Welzl E., 'Constructing the visibility graph for n line segments in $O(n^2)$ time'. *Information Processing Letters*, 20:167–171, 1985.

Triangulating a Convex Polygon with Small Number of Non-standard Bars

Extended Abstract

Yinfeng Xu[1], Wenqiang Dai[2], Naoki Katoh[3], and Makoto Ohsaki[3]

[1] School of Management, Xi'an Jiaotong University, Xi'an,710049, P.R. China
The State Key Lab for Manufacturing Systems Engineering, P.R. China
yfxu@mail.xjtu.edu.cn
[2] School of Management, Xi'an Jiaotong University, Xi'an, 710049, P.R. China
wqdai@mail.xjtu.edu.cn
[3] Department of Architecture and Architectural Engineering, Kyoto University
Kyotodaigaku-Katsura, Nishikyo, Kyoto 615-8540, Japan
{naoki,ohsaki}@archi.kyoto-u.ac.jp

Abstract. For a given convex polygon with inner angle no less than $\frac{2}{3}\pi$ and boundary edge bounded by $[l, \alpha l]$ for $1 \leq \alpha \leq 1.4$, where l is a given standard bar's length, we investigate the problem of triangulating the polygon using some Steiner points such that (i) the length of each edge in triangulation is bounded by $[\beta l, 2l]$, where β is a given constant and meets $0 < \beta < \frac{1}{2}$, and (ii) the number of non-standard bars in the triangulation is minimum. This problem is motivated by practical applications and has not been studied previously. In this paper, we present a heuristic to solve the above problem, which is based on the heuristic to generate a triangular mesh with more number of standard bars and shorter maximal edge length, and a process to make the length of each edge lower bounded. Our procedure is simple and easily implemented for this problem, and we prove that it has good performance guaranteed.

1 Introduction

Generating triangular meshes is one of the fundamental problems in computational geometry, and has been extensively studied; see e.g. the survey article by Bern and Eppstein[3]. From the view point of applications, it is important to impose geometric constraints on the shape of triangles in the obtained triangulation. Several measures of triangle quality, along with various algorithms to find optimal or near-optimal triangular meshes, have been reported [1, 2, 4–6, 10, 11].

For a given length l, we say that an edge is *standard bar* if its length is l while an edge is *non-standard bar* if its length is not. In this paper, we consider the problem of generating an edge bounded triangular mesh for a given convex polygon using some Steiner points so that the number of non-standard bars in the triangulation is minimized. The problem is similar to the one that finds an edge bounded triangulation where the number of standard bars is maximized, since a triangulation that achieves one of these objectives also does it well for

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 481–489, 2005.

the other, i.e., if a triangulation has increased the number of standard bars, it must decrease the number of non-standard bars, and vice versa.

This problem will be formalized as follows: we are given a convex polygon $\boldsymbol{P}$ with n vertices and a standard bar length l. It is assumed that every inner angle of $\boldsymbol{P}$ is no less than $\frac{2}{3}\pi$ and the length of every boundary edge is in the interval $[l, \alpha l]$, where $1 \leq \alpha \leq 1.4$. The objective is to generate a triangulation of $\boldsymbol{P}$ with every edge length is between βl and $2l$, and in a way that the number of non-standard bars is minimized (where β is a given constant and meets $0 < \beta < \frac{1}{2}$).

To the knowledge of the authors, the problem dealt with in the present paper has not been studied in the field of computational geometry. However, this problem appears in many practical applications. For example, in architecture design where the material is limited, to triangulate a convex polygon with some standard bars and less number of non-standard bars is often considered. The standard bar can be reused for many times, but the non-standard bars can't. Furthermore, from the practical point of view, there are also some constraints for the non-standard bars, for example, the length of the non-standard bar should be neither too long nor too short compared with the standard bar.

In this paper, we present a heuristic for constructing such a triangular mesh which is similar in simplicity and efficiency to standard algorithms for triangular mesh generation. The main idea is based upon the procedure to generate a triangulation with the number of standard bars as many as possible while the maximum edge length is short, and then upon the procedure to make every edge length bounded from below by a certain length. Our heuristic is capable of producing a triangulation with each edge bounded by $\left[\beta l, \max\{l+2\beta l, \frac{\sqrt{219}}{10}l+\beta l\}\right]$, which is contained in $[\beta l, 2l]$, and the number of non-standard bars is upper-bounded by $n + \left\lceil \frac{2}{\sqrt{3}}\alpha n \right\rceil$. Note that the number of interior Steiner points and triangles can go up to $O(n^2)$, so this $O(n)$ non-standard bars introduced by our heuristic are not large in number.

The rest of this paper is organized as follows. In section 2 we first provide a heuristic to obtain a triangulation $\mathcal{M}$ such that the number of standard bars in $\mathcal{M}$ is as many as possible, and that the maximum edge length in $\mathcal{M}$ is short. We examine the triangulation $\mathcal{M}$ in great detail. Especially, we find that the upper bound of each edge length is $\frac{\sqrt{219}}{10}l$, which is a tight bound, but the lower bound is not guaranteed. In section 3 we use an approach to make each edge length bounded from below by βl. Thus the "*new*" triangulation will meet the constraints of the problem. Finally the number of non-standard bars will be investigated in section 4 and section 5 gives some future works related to this paper.

2 A Triangulation with More Number of Standard Bars and Shorter Maximal Edge Length

In this section, we consider the problem of generating a triangulation for $\boldsymbol{P}$ with the number of standard bars maximized and the length of maximal edge in the

triangulation minimized. We shall give a heuristic for this problem and then show that the triangulation produced by our heuristic can be modified to give a good solution for the problem addressed in section 1.

The key idea behind the heuristic is to use the MinMax triangulation for a polygon. A MinMax edge triangulation stands for the triangulation that minimizes the maximum edge length in a triangulation over all possible triangulations of the given polygon.

Heuristic A

Step 1: Put $\boldsymbol{P}$ on the plane which is full of equilateral triangle lattice with edge length l.

Step 2: Let P' be the lattice set inside $\boldsymbol{P}$. Compute $B(P')$, where $B(P')$ denotes the boundary with lattice edges of P'.

Step 3: Let $CH(P)$ be the boundary of $\boldsymbol{P}$. Use P and $B(P')$ to triangulate the polygon region between $CH(P)$ and $B(P')$ under the MinMax edge criteria.

Let $\mathcal{M}$ be the triangulation obtained by the Heuristic A. Our aim is to present an upper bound of edge length in $\mathcal{M}$. To this end, firstly it is worth noting that, while using the Step 3 to obtain the MinMax edge triangulation, we must connect each vertex in $\boldsymbol{P}$ with its nearest vertex in $B(P')$ otherwise the maximal edge length will be longer. Thus, we define a polygon $\mathcal{A}$, which is a subgraph of $\mathcal{M}$, as follows:

Definition 1. *Let $e = (p, q)$ be a boundary edge of $\boldsymbol{P}$. Let p_1 and q_1 respectively, denote the lattice vertices nearest to p and q in $B(P')$. As polygon $\boldsymbol{P}$ is convex, pp_1 and qq_1 are on the same side of pq. We use the notation $\mathcal{A}$ to stand for the polygon composed of pq, pp_1, qq_1 and the path of lattice edges on $B(P')$ from p_1 to q_1.*

Polygon $\mathcal{A}$ may not be convex, we can not use the dynamic programming [8, 9] to obtain the MinMax edge triangulation of $\mathcal{A}$ in theory. However, as we will prove the number of edges in $\mathcal{A}$ is at most 6 in Lemma 4, the MinMax edge triangulation of $\mathcal{A}$ can be easily generated in practice.

From the above discussion, we can obtain the following lemma.

Lemma 1. *The maximum of the maximal edge length in the MinMax edge triangulation of all possible $\mathcal{A}$ is equal to the length of the maximum edge in $\mathcal{M}$.*

According to this lemma, in order to investigate the upper bound of edge length in $\mathcal{M}$, we only need to consider the maximum of maximal edge length in MinMax edge triangulation of $\mathcal{A}$. As $\mathcal{A}$ is for arbitrary boundary edge of $\boldsymbol{P}$, we turn to find the upper bound of the maximum edge in the MinMax edge triangulation of arbitrary $\mathcal{A}$.

Throughout this paper, we always use pq to denote the boundary edge in $\boldsymbol{P}$, and use p_1, q_1, respectively, to denote the lattice vertexes in $B(P')$ nearest to p and q. Sometimes we use the notation AB to directly denote the distance between point A and point B.

We begin with showing some properties of any polygon $\mathcal{A}$.

Lemma 2. *For any boundary edge pq of $\boldsymbol{P}$ in $\mathcal{A}$, there exists a vertex v on $B(P')$, such that either $0 \leq pv \leq l$ or $l < pv \leq \frac{2}{\sqrt{3}}l$. Furthermore, if pv satisfies $l < pv \leq \frac{2}{\sqrt{3}}l$ then the $\angle vpq$ in $\mathcal{A}$ is no more than $\frac{\pi}{2}$.*

Lemma 3. *Let $\mathcal{A}$ denote the polygon corresponding to the boundary edge L, and L_B be the lattice edge path on $\mathcal{A}$, and L_B^* be the length of the line segment connecting the two endpoints of L_B, then we have*

$$L_B^* \geq \frac{\sqrt{3}}{2} l \cdot n_L$$

where n_L denotes the number of lattice edges on L_B.

Lemma 4. *The number of edges in any polygon $\mathcal{A}$ is at most six.*

The following is a main theorem of this paper.

Theorem 1. *The maximum edge length in $\mathcal{M}$ is no more than $\frac{\sqrt{219}}{10}l$, and this upper bound is tight.*

Proof. We first summarize the proof. By Lemma 1, we may only need to investigate the upper bound of the maximum edge length in MinMax edge triangulation of $\mathcal{A}$. To this end, we show that for any case of $\mathcal{A}$, there exist a triangulation to make the length of maximum edge no more than $\frac{\sqrt{219}}{10}l$. Next for proving the tight upper bound, an actual $\mathcal{A}$ and its MinMax edge triangulation will be presented, whose maximum edge length in the triangulation is exactly $\frac{\sqrt{219}}{10}l$.

We now proceed with the details. If $p_1 = q_1$, that is, $\mathcal{A}$ is a triangle, the upper bound is αl. In the following we only consider the case that the number of edges in $\mathcal{A}$ is more than 3.

Recalling Lemma 4, $\mathcal{A}$ has at most six edges. The graph of $\mathcal{A}$ and its triangulation are just shown in Fig. 1, where $p_1A_1 = A_1A_2 = A_2q_1 = l$, and at the degenerate case, point p_1 may be equal to A_1, point q_1 coincides with A_2 and point A_1 may be equal to A_2. In the following we may only consider the non-degenerate cases since the degenerate one is a special case of non-degenerate cases. We draw the lines pA_1, pA_2 and qA_2 if $pA_2 \leq qA_1$ (see the left case of Fig. 1), or connect the line pA_1, qA_1 and qA_2 if $qA_1 < pA_2$ (see the right case of Fig. 1), to obtain the triangulation of $\mathcal{A}$. Without loss of generality, we assume $pA_2 \leq qA_1$ and only consider the left case of Fig. 1.

Firstly we have $pp_1 \leq pA_1$ and $qq_1 \leq qA_2$ by the definition of p_1 and q_1, so the possible maximal edge of triangulation is pq, pA_2, qA_2 or pA_1. We then distinguish the four different cases.

Case 1. The maximal edge is pq. For this case, the maximal edge length is αl and the upper bound is $1.4l$ as $\alpha \leq 1.4$.

Case 2. The maximal edge is pA_2. For this case, as $pA_2 \leq qA_1$, the length of pA_2 reaches its maximal length for the MinMax edge triangulation of $\mathcal{A}$, then the quadrilateral pqA_2A_1 is an isosceles trapezoid and the two edges pA_2 and

Fig. 1. Illustration used for the proof of Theorem 1: possible shapes of $\mathcal{A}$ and its triangulation. The left case is used for $pA_2 \leq qA_1$ and the right case is used for $qA_1 < pA_2$.

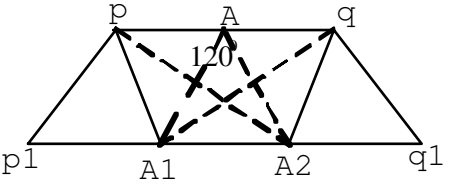


Fig. 2. Illustration used for the proof of Theorem 1, case 2.

qA_1 are the trapezoidal diagonals. In this case pq and A_1A_2 are parallel. So the length of pA_2 achieves the upper bound when the distance between pq and A_1A_2 reaches the maximum. The resulting $\mathcal{A}$ and its triangulation is shown in Fig. 2. According to cosine theorem in $\triangle ApA_2$, the upper bound of $pA_2$is

$$\left[(\frac{7}{10}l)^2 + l^2 - 2 \cdot \frac{7}{10}l \cdot l \cdot \cos(\frac{2}{3}\pi)\right]^{\frac{1}{2}} = \frac{\sqrt{219}}{10}l$$

Case 3. The maximal edge is qA_2. For this case, the upper bound is also $\frac{\sqrt{219}}{10}l$. The proof is done in the same manner as those given in case 2.

Case 4. The maximal edge is pA_1. For this case, we have $pA_1 \geq pq_1$ and $pA_1 \geq pA_2$ since pA_1 is the maximal edge. In the following we analyze the position of point "p" to show that this case does not happen.

Since $pA_1 \geq pq_1$, vertex p should belong to the left section of the midperpendicular line of p_1A_1. But vertex p also belongs to the right section of the midperpedicular line of A_1A_2 by $pA_1 \geq pA_2$. So vertex p must belong to the joint set of these two sections, that is, the polygon $\mathcal{A}$ must be like Fig. 3. However, in Fig. 3, vertex A is the nearest point to p, which contradicts the assumption that point p_1 is the point nearest to p. So pA_1 cannot be the maximal edge in $\mathcal{A}$.

Hence we have proved that the upper bound of maximum edge in MinMax edge triangulation of $\mathcal{A}$ is $\frac{\sqrt{219}}{10}l$, and from the Case 2 of proof, the tightness is obvious. □

By Theorem 1, we have obtained that the maximum edge length in triangulation $\mathcal{M}$ is no more than $\frac{\sqrt{219}}{10}l$. However, the lower bound of the edge length

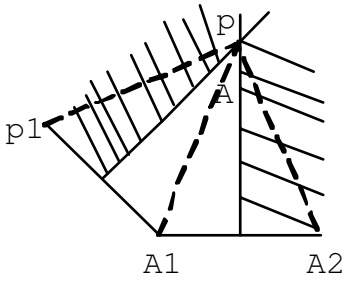


Fig. 3. Illustration used for the proof of Theorem 1, case 4.

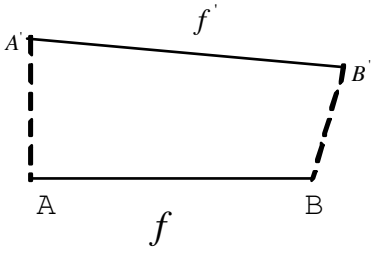


Fig. 4. Illustration used for the proof of Theorem 2, Case 3c.

has not be guaranteed in the obtained triangulation, i.e., some edges length in $\mathcal{M}$ may be very small. In the following we will consider the method to guarantee each edge length is no less than βl, where β is a given constant with $0 < \beta < \frac{1}{2}$.

3 A Triangulation with Edge Length No Less Than βl

We are now ready to show how triangulation $\mathcal{M}$ obtained by Heuristic A can be modified to give a solution for problem posed in the introduction. Theorem 1 implies the maximum edge length in $\mathcal{M}$ is bounded from above. Thus we only need to consider how to guarantee that edge lengths are bounded from below by βl. The key idea behind our heuristic is to simply contract those edges. (Note that we sometimes abuse f to denote the length of edge f.)

Heuristic B
Step 1-3: The same as Heuristic A. Denote the obtained triangulation by $\mathcal{M}$.
Step 4: For each edge f in $\mathcal{M}$, if $f < \beta l$ then one endpoint of f must be in P and the other must be in $B(P')$. Denote the endpoint of f in P by p and the endpoint in $B(P')$ by v, move v to p.

Note that for guaranteeing the existence of triangulation, we must let the "move" in Step 4 be "clockwise move" by the order of vertices of P. Let $\mathcal{N}$ denote the triangulation obtained by Heuristic B. The following theorem presents the length bound of edges in $\mathcal{N}$.

Theorem 2. *The edge lengths in triangulation $\mathcal{N}$ are in the interval*

$$\left[\beta l, \max\{l + 2\beta l, \frac{\sqrt{219}}{10} l + \beta l\}\right].$$

Proof. Since the lower bound βl is trivial, we need only to prove the upper bound. For each edge f in triangulation $\mathcal{M}$ of the polygon region between $CH(P)$ and $B(P')$, three cases are distinguished, according to the position of endpoints of f.

Case 1. Both of the two endpoints of f belong to P. For this case, edge f is an edge of $CH(P)$ and does not change by Heuristic B as $f \geq l$, thus $f \leq \alpha l \leq 1.4l$.

Case 2. One endpoint of f belongs to $\boldsymbol{P}$ and another endpoint of f belongs to $B(P')$.
Case2a: If the edge f do not change in $\mathcal{N}$, then we have $f \leq \frac{\sqrt{219}}{10} l$ by Theorem 1.
Case2b: Now assume the endpoint of edge f in $B(P')$ is moved, as the endpoint of f in $B(P')$ move to an vertex of $\boldsymbol{P}$, then the length of newly formed edges are bounded by $\frac{\sqrt{219}}{10} l + \beta l$ according to Theorem 1 and triangle inequality.

Case 3. Both of the two endpoints of f belong to $B(P')$.
Case3a: If edge f does not change in $\mathcal{N}$, then we have $f = l$.
Case3b: If only one endpoint of f changes in $\mathcal{N}$, the newly formed edges in $\mathcal{N}$ are no more than $\beta l + l$ according to triangle inequality.
Case3c: If both of the two endpoints of f moves in $\mathcal{N}$. See Fig. 4. Let edge f be AB, and let us assume vertex A moves to vertex A', vertex B moves to vertex B' and the newly formed edge f' is denoted by $A'B'$. The edges f, f', $A'A$ and $B'B$ formes a quadrangle. We have $AA' < \beta l$, $BB' < \beta l$ and $f = l$, thus triangle inequality gives $f' < A'A + AB + BB' < l + 2\beta l$.

Thus, the edge lengths of $\mathcal{N}$ are upper bounded by $\max\{l + 2\beta l, \frac{\sqrt{219}}{10} l + \beta l, \beta l + l, 1.4l, l\} = \max\{l + 2\beta l, \frac{\sqrt{219}}{10} l + \beta l\}$ and the theorem is proved. □

By Theorem 2 and $l + 2\beta l \leq 2l$, $\frac{\sqrt{219}}{10} + \beta l < 2l$, the Heuristic B is actually capable of generating the triangulation with all edges bounded by $[\beta l, 2l]$, thus meet the need of the primal problem.

4 On the Number of Non-standard Bars

To estimate the performance of $\mathcal{N}$, we consider the final procedure shown in Heuristic B. Since the number of edges in $\mathcal{N}$ is no more than the number of edges in $\mathcal{M}$, the number of non-standard bars is bounded by the number of edges in the triangulation of the region between $\boldsymbol{P}$ and $B(P')$.

Lemma 5. *The number of lattice edges on $B(P')$ is bounded by $\left\lceil \frac{2}{\sqrt{3}} \alpha \cdot n \right\rceil$.*

Lemma 6. *The number of edges on $CH(B(P'))$ is bounded by $\left\lceil \frac{2}{\sqrt{3}} \alpha \cdot n \right\rceil$.*

Theorem 3. *The number of edges in a triangulation of the region between $\boldsymbol{P}$ and $B(P')$ is bounded by $n + \left\lceil \frac{2}{\sqrt{3}} \alpha \cdot n \right\rceil$.*

Proof. Let S_1 denote the point set of P and S_2 denote the point set of P'. The Eulerian relation [7] for planar graph implies the following equalities:

$$|T(S_1 \cup S_2)| = 3|S_1 \cup S_2| - |CH(S_1 \cup S_2)| - 3$$
$$|T(S_2)| = 3|S_2| - |CH(S_2)| - 3$$

where $|T(S_1 \cup S_2)|$ and $|T(S_2)|$ denote the number of edges in triangulation $T(S_1 \cup S_2)$ and triangulation $T(S_2)$, respectively, $|S_1 \cup S_2|$ and $|S_2|$ denote the number of points in $S_1 \cup S_2$ and S_2, respectively, and $CH(S_1 \cup S_2)$ and $CH(S_2)$ are the number of edges in convex hull of $S_1 \cup S_2$ and S_2, respectively.

We have

$$\begin{aligned}
&|S_1 \cup S_2| = |S_1| + |S_2|,\\
&|CH(S_1 \cup S_2)| = |P| = n,\\
&|CH(S_2)| = |CH(B(P')| \leq \left\lceil \tfrac{2}{\sqrt{3}}\alpha \cdot n \right\rceil .
\end{aligned}$$

where the first equality uses $S_1 \cap S_2 = \emptyset$ and the final inequality uses Lemma 6. Then

$$\begin{aligned}
|T(S_1 \cup S_2)| - |T(S_2)| &= 3|S_1 \cup S_2| - |CH(S_1 \cup S_2)| - 3|S_2| + |CH(S_2)|\\
&= 3|S_1| - n + |CH(S_2)|\\
&= 2n + |CH(S_2)|\\
&\leq 2n + \left\lceil \tfrac{2}{\sqrt{3}}\alpha n \right\rceil .
\end{aligned}$$

where the third step uses the fact that the number of points in set S_1 is equal to n.

Thus we finish the proof by investigating that the number of edges in triangulation of the region between $\boldsymbol{P}$ and $B(P')$ is just $|T(S_1 \cup S_2)| - |T(S_2)|$ minus the number of edges of $\boldsymbol{P}$. □

Remark 1. If $B(P')$ is a convex polygon, then the number of lattice edges on $B(P')$ is bounded by $\lceil \alpha n \rceil$, and the number of edges in a triangulation of the region between P and $B(P')$ is bounded by $n + \lceil \alpha n \rceil$.

5 Conclusion and Future Work

In this paper, we have presented heuristics to generate a triangular mesh with the number of standard bars as many as possible. An interesting open problem is to investigate whether we can refine this procedure to obtain better results. What is more, our problem is a simple form of the following general problem:

For given real numbers $\alpha \leq \beta \leq \gamma$, and a convex polygon $\boldsymbol{P}$, how can we find a Steiner triangulation, $T(P)$, of P such that the length of inner edge in $T(P)$ is in the interval $[\alpha, \gamma]$ and the number of edges with edge length different from β is minimum?

All results given in this paper hold for polygon with boundary edge bounded by $[l, \alpha l]$ for $1 \leq \alpha \leq 1.4$, what is the largest value for α to let our results hold is still an open problem.

Acknowledgements

This research is supported by NSF of China under Grants 10371094 and 70471035. We also gratefully acknowledge a number of valuable comments and suggestions given by the anonymous referees.

References

1. F. Aurenhammer, N. Katoh, H. Kojima, M. Ohsaki and Y. F. Xu, Approximating uniform triangular meshes in polygons, *Theoretical Computer Science*, 289(2002), pp 879-895
2. M. Bern, D. Dobkin, D. Eppstein, Triangulating polygons without large angles, *Internat. J. Comput. Geom. Appl.* 5(1995) 171-192.
3. M. Bern, D. Eppstein, Mesh generation and optimal triangulation, in: D. -Z. Du (Ed.), *Computing in Euclidean Geometry*, World Scientific, Singapore, 1992, 47-123.
4. M. Bern, D. Eppstein, J. R. Gilbert, Provably good mesh generation, *J. Comput. System Sci.* 48 (1994) 384-409.
5. M. Bern, S. Mitchell, J. Ruppert, Linear-size nonobtuse triangulation of polygons, Proc. 10th Ann. ACM Symp. on Computational Geometry, 1994, 221-230.
6. P. Chew, Guaranteed-quality mesh generation for curved surfaces, Proc. 9th Ann. ACM Symp. on Computational Geometry, 1993, 274-280.
7. H. Edelsbrunner, *Algorithms in Combinatiorial Geometry*, EATCS Monographs on Theoretical Cmputer Science 10, Springer-Verlag, 1987
8. P. D. Gilbert, *New results in planar triangulation*, report R-850, Coordinated Science Laboratory, University of Illinois, 1979.
9. G. T. Klincsek, Minimal triangulations of polygonal domains, *Ann. Discrete Math*, 9, 127-128, 1980
10. E. Melisseratos, D. Souvaine, Coping with inconsistencies: a new approach to produce quality triangulations of polygonal domains with holes, Proc. 8th Ann. ACM Symp. on Computational Geometry, 1992, 202-211.
11. J. Ruppert, A Delaunay refinement algorithm for quality 2-dimensional mesh generation, *J. Algorithms* 18 (1995) 548-585.

A PTAS for a Disc Covering Problem Using Width-Bounded Separators*

Zhixiang Chen[1], Bin Fu[2,3], Yong Tang[4], and Binhai Zhu[5]

[1] Dept. of Computer Science,University of Texas -Pan American, TX 78539, USA
chen@cs.panam.edu
[2] Dept. of Computer Science, University of New Orleans, LA 70148, USA
[3] Research Institute for Children, 200 Henry Clay Avenue, New Orleans, LA 70118
fu@cs.uno.edu
[4] Dept. of Computer Science, Sun Yat-Sen University, Guangzhou 510275, China
issty@zsu.edu.cn
[5] Dept. of Computer Science, Montana State University, MT 59717-3880, USA
bhz@cs.montana.edu

Abstract. In this paper, we study the following disc covering problem: Given a set of discs of various radii on the plane, find a subset of discs to maximize the area covered by exactly one disc. This problem originates from the application in digital halftoning, with the best known approximation factor being 5.83 [2]. We show that if the maximum radius is no more than a constant times the minimum radius, then there exists a polynomial time approximation scheme. Our techniques are based on the width-bounded geometric separator recently developed in [5, 6].

1 Introduction

In real life we are always dealing with the problem of mixed technology; for instance maintaining COBOL and JAVA compilers at the same time. It is also not uncommon that sometimes we have to print some colored fancy images onto a black/white tone printer. Digital-halftoning is exactly such a technology, it converts a continuous, possibly colored image into a binary image [11, 12]. In the cluster-dot halftoning, dots form clusters whose sizes are determined by their corresponding intensity level. Given a continuous-tone image, one computes spatial frequency distribution by Laplacian. Each grid point is then assigned a disc of radius reflecting the Laplacian value at the corresponding position. This results in a set of discs of different radii. The problem is then to find a subset of discs to maximize the area that belongs to exactly one disc.

We study the approximation algorithm for the above disc covering problem with applications in digital halftoning [2, 3, 11, 12, 14]. Given a set of discs of various radii, find a subset of discs from them to maximize the area covered by exactly one disc. This seems computationally hard although there is not yet a proof about NP-hardness. We show that if the maximum radius is no more than

* This research is supported by Louisiana Board of Regents fund under contract number LEQSF(2004-07)-RD-A-35.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 490–503, 2005.

a constant times the minimum radius, there exists a polynomial time approximation scheme. If the centers of the discs are at the grid points and the radii are between two positive constants, there exists a constant factor approximation which runs in almost linear time.

In [2], a polynomial time approximation algorithm was designed with approximation ratio 5.83. In their algorithm, no condition is specified that the maximum radius is no more than a constant times the minimum radius. However, the empirical data used in [2] shows that not only such a constant stands, it is also always relatively small (i.e., 3-5). We believe that this assumption is practically reasonable since each disc reflects the intensity level of a local point.

Geometric separator has applications in many problems. It plays important role when we develop divide and conquer algorithm for geometric problems. Lipton and Tarjan [9] presented the well known geometric separator for planar graphs. They proved that every n-vertex planar graph has at most $\sqrt{8n}$ vertices whose removal separates the graph into two disconnected parts of size at most $\frac{2}{3}n$. Their $\frac{2}{3}$-separator was improved to $\sqrt{6n}$ by Djidjev [4], $\sqrt{5n}$ by Gazit [7], and $\sqrt{4.5n}$ by Alon, Seymour and Thomas [1]. Spielman and Teng [16] showed a $\frac{3}{4}$-separator with size $1.82\sqrt{n}$ for planar graph.

Some other forms of the separators were studied in [10, 15]. They let each input point be covered by a regular geometric object such as circle, rectangle, etc. If every point on the plane is covered by at most k objects, it is called k-thick. Some separators of size $c\cdot\sqrt{k\cdot n}$ were proved in [10, 15], where c is a constant. Fu and Wang [6] developed a method for deriving sharper upper bound separator for grid points via controlling the distance to the separator line. They proved that for a set of n grid points on the plane, there is a separator that has $\leq 1.129\sqrt{n}$ points and each side has $\leq \frac{2}{3}n$ points. Fu [5] introduced the concept of width-bounded geometric separator and applied it to a class of NP-complete geometric problems to improve their computational time from $n^{O(\sqrt{n})}$ to $2^{O(\sqrt{n})}$. In this paper we use the width-bounded geometric separator to develop a polynomial time approximation scheme for the halftoning problem.

2 Separators on the Plane

Definition 1. *For two points p_1, p_2 in the plane R^2,* $\mathrm{dist}(p_1, p_2)$ *is the Euclidean distance between p_1 and p_2. For a set $A \subseteq R^2$,* $\mathrm{dist}(p_1, A) = \min_{q\in A}\mathrm{dist}(p_1, q)$. *Let P be a set of points on the plane, and $w > 0$ be a constant. A w-wide-separator is determined by a line L, called the center line of the separator, on the plane. It has two measurements for its quality of separation: (1)* $\mathrm{balance}(L, P) = \frac{\max(|P_1|,|P_2|)}{|P|}$, *where P_1 and P_2 are the two subsets of P on the two sides of L; and (2)*$\mathrm{measure}(L, P, \frac{w}{2})$, *which is the number of elements of P with distance $\leq \frac{w}{2}$ to L. The w-width separator area is all points with distance $\leq \frac{w}{2}$ to L. For constants $0 < b_0 < 1$, $z_0 \geq 0$, $w \geq 0$, and a set of n grid points P on the plane, a (b_0, z_0)-w-width-separator (for P) is a w-width separator L with* $\mathrm{balance}(L, P) \leq b_0$ *and* $\mathrm{measure}(L, P, \frac{w}{2}) \leq \frac{z_0 w}{2}\sqrt{n}$.

From the definition of width-bounded separator, its quality is measured by two numbers. One measures the balance of the separation. A well balanced sepa-

rator can reduce the problem size efficiently during the application to divide and conquer algorithm. This brings that the algorithm runs in a polynomial time. The other number measures the number of points inside the separator area. The small number of points in the separator area ($O(\sqrt{n})$) is used to control the accuracy of our approximation algorithm.

Theorem 1. *[5, 6] Let constant $w > 0$ be a constant and $\delta > 0$ be a small constant. Let P be a set of n grid points. Then there is an $O(n^3)$ time algorithm that finds a separator line L such that each side of L has $\leq \frac{2}{3}n$ points from P, and the number of points of P with distance $\leq w$ to L is $\leq (\frac{4}{\sqrt{\pi}} + \delta)w \cdot \sqrt{n}$ for all large n.*

3 The Approximation Scheme

Definition 2. *For constant $c > 0$, the input is a set of discs $D_1, \cdots, D_n$ on the plane with $r(D_i) \leq c \cdot r(D_j)$ for all $1 \leq i, j \leq n$, where $r(D_i)$ is the radius of D_i. The H_c problem P is to find a subset $Q \subseteq P$ with the maximal area covered by exactly one disc in Q. Define* opt(P) *to be the subset of discs of P in an optimal solution. The H'_c problem P is a special H_c problem such that the distance between every pair of disc centers in P is at least $c' \times r(D_i)$ for any D_i in the P, where $c' > 0$ is a fixed constant. This problem studied by [2] requires that every center is a grid point. If the radii are between two positive constants then it is covered by our definition. For a grid point $p = (i, j)$ (i and j are integers) on the plane, define* grid$(p) = \{(x, y) | i - \frac{1}{2} \leq x < i + \frac{1}{2}, j - \frac{1}{2} < y \leq j + \frac{1}{2}\}$, *which is a half close and half open 1×1 square. The net $g(P)$ for a H_c problem P is a set of grid points such that (1) for each point $p \in g(P)$,* grid(p) *contains the center for some disc in P; and (2) for each disc D of P,* center$(D) \in$ grid(p) *for some point p in $g(P)$, where* center(D) *is the center point of disc D. For a set of discs Q on the plane, define $s(Q)$ to be the size of the area covered by exactly one disc in Q.*

In the theorem below, the function $f_P(e)$ controls the number of disc centers in the area with e grid points. The purpose of the function f_P is to unify the algorithms for both H_c and H'_c problems. For an H_c problem, $f_P(O(1))$ is up to $|P|$, but for an H'_c problem, $f_P(O(1)) = O(1)$. Our approximation scheme depends on the algorithm to find the width-bounded separator for a set of grid points on the plane. Theorem 1 gives $O(n^3)$ time algorithm for finding the width bounded separator. An $O(n(\log n)^4)$ time randomized algorithm for finding separator is presented at section 4. Our Theorem 2 shows how the time of our approximation algorithm depends on the time for the separator detection. This is why it assumes there exists an $O(n^a(\log n)^b)$ time algorithm for finding separator, where a, b are constants.

Theorem 2. *Let $0 < b_0 < 1$, $0 \leq z_0$, and $0 < \epsilon$ be constants. Let P be an H_c problem and f_P be an non-decreasing function from N to N such that $|Q| \leq f_P(|g(Q)|)$ for every $Q \subseteq P$. Assume that there exists an $O(n^a(\log n)^b)$ time*

algorithm for computing the (b_0, z_0)-$O(1)$-*width-bounded separator for some constants* $a \geq 1$ *and* $b \geq 0$. *Then there exists an* $O(f_P\left(\frac{E_1}{\epsilon^{\frac{1}{1-\alpha}}}\right)^{\frac{E_2}{\epsilon^{\frac{1}{1-\alpha}}}} n^a(\log n)^{b+1})$ *time approximation algorithm to output* $Q \subseteq P$ *with* $s(Q) \geq (1-\epsilon)s(\text{opt}(P))$, *where* $\alpha = 0.6$, E_1 *and* E_2 *are constants.*

Proof. We first give an overview about our method. Assume the minimum radius of the input discs is 1. The radius of every disc of P is $\leq c$. For a set of discs $P = \{D_1, \cdots, D_n\}$ on the plane, the net $g(P)$ shows that the optimal solution of P has $\Omega(|g(P)|)$. Apply a separator with width $\geq 2c$. The discs on the different sides of the separator do not intersect each other. The two sub-problems on the left and right sides of the separator can solved independently. Our separator can control there are only $O(\sqrt{|g(P)|})$ points from $g(P)$ to stay in the separator area. The discs on the separator area only affect the overall solution by $O(\sqrt{|g(P)|})$, which does not affect its total accuracy much. Our algorithm is based on such a divide and conquer approach by using width-bounded geometric separator.

Let $\epsilon > 0$ be a constant that determines the accuracy of our approximation algorithm. Let P be the H_c problem, which consists of a set of discs on the plane. Select some constants: $w_0 = c + \frac{\sqrt{2}}{2}$, $\delta = 0.01$, $b_1 = 1 - b_0$, $\delta_1 = \min(0.08, \frac{b_1}{4})$, $c_2 = \pi(\frac{\sqrt{2}}{2} + c)^2$ and $c_3 = \frac{1}{\pi(2\sqrt{2}+2c+\frac{\sqrt{2}}{2})^2}$, $\alpha = 0.6$, and e_1 is a constant that satisfies the inequalities:

$$\frac{z_0 w_0}{\sqrt{e_1}} \leq \delta_1, \tag{1}$$

$$\epsilon(c_3(b_1 - 2\delta_1)e_1) > ((b_1 - 2\delta_1)e_1)^\alpha, \text{ and} \tag{2}$$

$$c_2 z_0 w_0 \sqrt{e_1} \leq \delta_1 e_1^\alpha. \tag{3}$$

We can choose constant E_1 big enough and let $e_1 = \frac{E_1}{\epsilon^{\frac{1}{1-\alpha}}}$. Then e_1 satisfies the conditions (1)-(3).

Algorithm

Input: a set of discs $P = \{D_1, \cdots, D_n\}$ on the plane

Output: A subset $A(P) \subseteq P$ with $s(A(P)) \geq (1-\epsilon)s(\text{opt}(P))$.

If $|g(P)| \leq e_1$, then find $A(P) = \text{opt}(P)$ using the brute-force method and return $A(P)$.

Find a $2w_0$-width separator center line L for $g(P)$ such that balance$(L, g(P)) \leq b_0$ and measure$(L, g(P), w_0) \leq z_0 w_0 \sqrt{|g(P)|}$ (see Theorem 1).

Let P_0 be all the discs D of P with dist(center$(D), L) \leq c$.

Let P_1 be all the discs D of centers on the one side of the separator and dist(center$(D), L) > c$.

Let P_2 be all the discs D of centers on the other side of the separator and dist(center$(D), L) > c$.

Solve P_1 to get the approximate solution $A(P_1)$.

Solve P_2 to get the approximate solution $A(P_2)$.

Merge the solutions for P_1 and P_2 to output $A(P) = A(P_1) \cup A(P_2)$.

End of Algorithm

Lemma 1. *Every $\delta \times \delta$-square has $\leq K$ disc centers from P in the optimal solution, where $K = 20$.*

Proof. Assume that $\text{opt}(P)$ has more than K centers in a $\delta \times \delta$ square. Let $\eta = \frac{c-1}{K}$. All of the K radii are in the range $[1, c]$, which can be partitioned into the union of K intervals of format $[1 + (i-1)\eta, 1 + i\eta]$ for $i = 1, 2, \cdots, K$. At least two discs in $\text{opt}(P)$ have radii in an interval $[1 + (i-1)\eta, 1 + i\eta]$ for some $i \in \{1, 2, \cdots, K\}$.

Let C_1 and C_2 be the two discs (in $\text{opt}(P)$) whose centers are in the same $\delta \times \delta$-square and radii are in the same interval $[1 + (i-1)\eta, 1 + i\eta]$. For a region R, let $v(R)$ be the area size of R. The two centers of discs C_1 and C_2 are close. So are their radii. It is easy to verify that $v(C_1 - C_2) \leq 0.2 \cdot v(C_1)$ and $v(C_2 - C_1) \leq 0.2 \cdot v(C_1)$. Let $R_0 \subseteq C_1$ be the maximal sub-region of C_1 such that every point in R_0 is covered by exactly one disc in $\text{opt}(P) - \{C_1, C_2\}$. We check the following two cases:

Case 1: $v(R_0) \geq 0.6 \cdot v(C_1)$. Since C_1 and C_2 are in $\text{opt}(P)$, every point in $C_1 \cap C_2$ is covered by at least two discs in $\text{opt}(P)$. We have that $s(\text{opt}(P) - \{C_1, C_2\}) \geq s(\text{opt}(P)) + v(R_0) - v(C_1 - C_2) - v(C_2 - C_1) \geq s(\text{opt}(P)) + 0.6v(C_1) - 0.2v(C_1) - 0.2v(C_1) > s(\text{opt}(P))$. This contradicts that $\text{opt}(P)$ is the optimal solution.

Case 2: $v(R_0) < 0.6 \cdot v(C_1)$. We have that $s(\text{opt}(P) - \{C_2\}) \geq s(\text{opt}(P)) + (v(C_1) - v(R_0)) - v(C_2 - C_1) \geq s(\text{opt}(P)) + 0.4v(C_1) - 0.2v(C_1) > s(\text{opt}(P))$. This is also a contradiction.

Lemma 2. *Let P be a H_c problem. Then (1) $s(\text{opt}(P)) \leq c_2|g(P)|$, and (2) $c_3|g(P)| \leq s(\text{opt}(P))$.*

Proof. (1) For every point q in a disc of P, there is a grid point $p \in g(P)$ with $\text{dist}(p, q) \leq \frac{\sqrt{2}}{2} + c$. Therefore, $s(\text{opt}(P)) \leq |g(P)|\pi(\frac{\sqrt{2}}{2} + c)^2$. (2) We prove this by induction. It is clearly true when $|g(P)| \leq 1$. Assume it is true for $|g(P)| < k$. Let $k = |g(P)|$. Select a grid point $p \in g(P)$. Let M_1 be the set of all discs D in P such that $\text{center}(D) \in \text{grid}(p)$. Let M_2 be the set of all discs D' in P such that $D' \cap D \neq \emptyset$ for some $D \in M_1$. Let $P' = P - M_1 \cup M_2$. The problem P is adjusted to the problem P'. For every point $p' \in g(P) - g(P')$, $\text{dist}(p, p') \leq 2(\frac{\sqrt{2}}{2} + c)$. The number of grid points with distance $\leq 2(\frac{\sqrt{2}}{2} + c)$ to p is $\leq \pi(2\sqrt{2} + 2c + \frac{\sqrt{2}}{2})^2 = \frac{1}{c_3}$. So, we have $|g(P')| \geq |g(P)| - \frac{1}{c_3}$. For $D \in M_1$, $s(\text{opt}(P)) \geq s(\{D\} \cup \text{opt}(P')) \geq s(\text{opt}(P')) + \pi \geq c_3|g(P')| + \pi \geq c_3(|g(P)| - \frac{1}{c_3}) + \pi \geq c_3|g(P)|$.

Lemma 3. *The algorithm has solution with $s(A(P)) \geq (1 - \epsilon)s(\text{opt}(P)) + (|g(P)|)^\alpha$ if $|g(P)| \geq (b_1 - 2\delta_1)e_1$.*

Proof. We prove by induction. If $(b_1 - 2\delta_1)e_1 \leq |g(P)| \leq e_1$, $s(A(P)) = s(\text{opt}(P)) \geq (1 - \epsilon)s(\text{opt}(P)) + (g(|P|))^\alpha$ by the inequality (2) and part (2) of Lemma 2. Assume that $|g(P)| \geq e_1$ and let L be the center line of the $2w_0$-width separator for $g(P)$. Let P_0, P_1 and P_2 are the sub-problems derived from P in the algorithm.

It is easy to see that $s(\text{opt}(P)) \leq s(\text{opt}(P_1)) + s(\text{opt}(P_2)) + s(\text{opt}(P_0))$. Therefore, $s(\text{opt}(P_1)) + s(\text{opt}(P_2)) \geq s(\text{opt}(P)) - s(\text{opt}(P_0))$. Clearly, $g(P_0)$ is

the subset of $g(P)$ with distance $\leq (c + \frac{\sqrt{2}}{2}) \leq w_0$ to L. Therefore, $|g(P_0)| \leq z_0 w_0 \sqrt{|g(P)|}$. By Lemma 2, $s(\text{opt}(P_0)) \leq c_2 |g(P_0)| \leq c_2 \cdot z_0 w_0 \sqrt{|g(P)|}$.

Let G_1 (G_2) be the set of grid points of $g(P)$ on the left (right resp.) of the center line L of the separator. Let S be the set of grid points of $g(P)$ inside the separator area (with distance $\leq w_0$ to L). Thus, $|S| \leq z_0 w_0 \sqrt{|g(P)|}$. We have $|G_1|, |G_2| \leq b_0 |g(P)|$ (Notice that b_0 is the balance upper bound for the separator).

For each $p \in g(P_1)$, there exists a disc $D \in P_1$ with $\text{dist}(p, \text{center}(D)) \leq \frac{\sqrt{2}}{2}$. Since center$(D)$ is on one side of L, p can not stay on the other side of L and has distance more than $\frac{\sqrt{2}}{2} (\leq w_0)$ to L. Thus, $p \in G_1 \cup S$. Therefore, $g(P_1) \subseteq G_1 \cup S$. For a grid point $q \in G_1 - S$, there exists $D \in P$ such that center$(D) \in \text{grid}(q)$. Since q has distance $> w_0$ to L, center(D) has distance $> w_0 - \frac{\sqrt{2}}{2} = c$ to L. So, $D \notin P_0 \cup P_2$, which implies $D \in P_1$. We have $G_1 - S \subseteq g(P_1)$. We have proven that $G_1 - S \subseteq g(P_1) \subseteq G_1 \cup S$. Similarly, $G_2 - S \subseteq g(P_2) \subseteq G_2 \cup S$. The set $G_1 \cup G_2$ contains all of the grid points in $g(P)$ except those in the line L. So, $g(P) \subseteq G_1 \cup G_2 \cup S$.

Thus, we have the following inequalities: $|g(P)| \leq |G_1| + |G_2| + |S|$; $|G_1| \leq b_0 |g(P)|$; $|G_2| \leq b_0 |g(P)|$; $|G_1| - |S| \leq |g(P_1)| \leq |G_1| + |S|$; and $|G_2| - |S| \leq |g(P_2)| \leq |G_2| + |S|$. Since $\frac{|S|}{|g(P)|} \leq \frac{z_0 w_0 \sqrt{|g(P)|}}{|g(P)|} \leq \frac{z_0 w_0}{\sqrt{|g(P)|}} \leq \frac{z_0 w_0}{\sqrt{e_1}} \leq \delta_1$ (by (1)), we have

$$|g(P_1)| \geq (b_1 - 2\delta_1)|g(P)| \tag{4}$$

$$|g(P_2)| \geq (b_1 - 2\delta_1)|g(P)| \tag{5}$$

$$|g(P_1)| + |g(P_2)| \geq (1 - 3\delta_1)|g(P)| \tag{6}$$

By our inductive assumption, (4) and (5), $s(A(P_1)) \geq (1-\epsilon) s(\text{opt}(P_1)) + (|g(P_2)|)^\alpha$, and $s(A(P_2)) \geq (1-\epsilon) s(\text{opt}(P_2)) + (|g(P_2)|)^\alpha$. Let $g(P_1)| = \beta_1 |g(P)|$ and $|g(P_2)| = \beta_2 |g(P)|$. We have $\beta_1 + \beta_2 \geq 1 - 3\delta_1$ and $\beta_1, \beta_2 \geq b_1 - 2\delta_1$. By the standard method in calculus, $\beta_1^\alpha + \beta_2^\alpha$ is minimal when $\beta_1 = \beta_2 = \frac{1-3\delta_1}{2}$. So, $\beta_1^\alpha + \beta_2^\alpha \geq 2(\frac{1-3\delta_1}{2})^\alpha = 2^{1-\alpha}(1-3\delta_1)^\alpha > 2^{1-\alpha}(1-3\delta_1\alpha) > 1.12 > 1 + \delta_1$. So, $|g(P_1)|^\alpha + |g(P_2)|^\alpha > (1+\delta_1)|g(P)|^\alpha$. Since $|g(P)| \geq e_1$, $|g(P_1)|^\alpha + |g(P_2)|^\alpha - c_2 z_0 w_0 \sqrt{|g(P)|} > |g(P)|^\alpha$ by inequality (3). Therefore, $s(A(P)) \geq s(A(P_1)) + s(A(P_2)) \geq (1-\epsilon)(s(\text{opt}(P_1)) + s(\text{opt}(P_2)) + (|g(P_1)|)^\alpha + (|g(P_2)|)^\alpha \geq (1-\epsilon)(s(\text{opt}(P)) - s(\text{opt}(P_0)) + (|g(P_1)|)^\alpha + (|g(P_2)|)^\alpha \geq (1-\epsilon) s(\text{opt}(P)) - c_2 \cdot z_0 \cdot w_0 \sqrt{|g(P)|} + (|g(P_1)|)^\alpha + (|g(P_2)|)^\alpha \geq (1-\epsilon) s(\text{opt}(P)) + (|g(P)|)^\alpha$.

Lemma 4. *The optimal solution* opt(P) *can be computed in* $O(|P|^{\frac{2|g(P)|K}{\delta^2}})$ *time by the brute force method.*

Proof. For each disc D in P, center$(D) \in \text{grid}(q)$ for some $q \in g(P)$. All centers of discs in P stay in the area of size $\leq |g(P)|$. By Lemma 1, opt(P) has $\leq \frac{2|g(P)|K}{\delta^2}$ discs. The lemma follows since each disc in the optimal solution has $\leq |P|$ choices.

Lemma 5. *The total time of the algorithm is* $O(M \cdot n^a (\log n)^{b+1})$, *where* $M = f_P(e_1)^{\frac{2e_1 K}{\delta^2}}$.

Proof. Let $m = |g(P)|$ and $T(m)$ be the time complexity of the algorithm. Clearly, $m \leq n$, where $n = |P|$. Assume that C_4 is a positive constant such that finding the separator takes $\leq C_4 m^a (\log m)^b$ steps. By Lemma 4 and $|P| \leq f(|g(P)|)$, $T(m) \leq M$ for $m \leq e_1$. We have $T(m) \leq C_5 MT(\gamma_1 m) + C_5 MT(\gamma_2 m) + C_4 m^a (\log m)^b$, where $0 \leq \gamma_1, \gamma_2 \leq b_0$, $\gamma_1 + \gamma_2 \leq 1$, and C_5 is a constant that is selected big enough so that we have following:

$$\begin{aligned} T(m) &\leq C_5 MT(\gamma_1 m) + C_5 MT(\gamma_2 m) + C_4 m^a (\log m)^b \\ &\leq C_5 M(\gamma_1 m)^a (\log \gamma_1 m)^{b+1} + C_5 M(\gamma_2 m)^a (\log \gamma_2 m)^{b+1} + C_4 m^a (\log m)^b \\ &\leq C_5 M m^a (\log m)^{b+1}. \end{aligned}$$

Since $e_1 = \frac{E_1}{\epsilon^{\frac{1}{1-\alpha}}}$, we let $E_2 = \frac{2E_1 K}{\delta^2}$. The theorem follows from Lemma 5 and Lemma 3.

Corollary 1. *Let $0 < b_0 < 1$, $0 \leq z_0$, and $0 < \epsilon$ be constants. Let P be an H_c problem. Assume that there exists an $O(n^a (\log n)^b)$ time algorithm for computing the (b_0, z_0)-$O(1)$-width-bounded separator with constants $a \geq 1$ and $b \geq 0$. Then there exists an $O((n^{\frac{E_2}{\epsilon^{\frac{1}{1-\alpha}}}}) n^a (\log n)^{b+1})$ time approximation algorithm to output $Q \subseteq P$ with $s(Q) \geq (1-\epsilon) s(\mathrm{opt}(P))$, where $\alpha = 0.6$, and E_2 is a constant.*

Corollary 2. *Let $0 < b_0 < 1$, $0 \leq z_0$, and $0 < \epsilon$ be constants. Let P be an H'_c problem. Assume that there exists an $O(n^a (\log n)^b)$ time algorithm for computing the (b_0, z_0)-$O(1)$-width-bounded separator with constants $a \geq 1$ and $b \geq 0$. Then there exists an $O(n^a (\log n)^{b+1})$ time approximation algorithm to output $Q \subseteq P$ with $s(Q) \geq (1-\epsilon) s(\mathrm{opt}(P))$.*

4 A Randomized Algorithm to Find the Separator

From corollary 1 and corollary 2, the separator algorithm affects the speed of our approximation. In this section, we will give an $O(n(\log n)^4)$-time randomized algorithm for finding the width-bounded separator on the plane. We will use the following well known fact that can be easily derived from Helly theorem (see [8, 13]).

Lemma 6. *For an n-element set P in d-dimensional space, there is a point q with the property that any half-space that does not contain q, covers at most $\frac{d}{d+1} n$ elements of P. Such a point q is called a centerpoint of P. The point q is called $\frac{2}{3}$-center at the case $d = 2$.*

Let $c \geq 3$ be a constant. For a set of n grid points P, we first sort them by their x-coordinates. Now let $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$ be all points of P and their x-coordinates are sorted by increasing order: $x_1 \leq x_2 \leq \cdots \leq x_n$. Let $i_1, \cdots, i_k$ be the positions such that $|x_{i_j} - x_{i_j+1}| \geq n^{c-1}$ $(i = 1, \cdots, k)$. Partition P into $P_1, \cdots, P_k$, where $P_t = \{(x_j, y_j) | i_t \leq j < i_{t+1}\}) (t = 1, 2, \cdots, k)$. Since $|P| = n$, $|x_{j_1} - x_{j_2}| \leq n \cdot n^{c-1} = n^c$ for every two points $(x_{j_1}, y_{j_1}), (x_{j_2}, y_{j_2})$

in the same set P_t. On the other hand, $|x_{j_1} - x_{j_2}| \geq n^{c-1}$ for every two points $(x_{j_1}, y_{j_1}), (x_{j_2}, y_{j_2})$ in the different sets P_{t_1} and P_{t_2}, respectively. We act the same on each P_i by their y-coordinates. Then P is partitioned into $\cup_{i,j} P_{i,j}$ such that each $P_{i,j}$ is inside an square of size $n^c \times n^c$, and the distance between two points in two different subsets sets P_{i_1,j_1} and P_{i_2,j_2} is at least n^{c-1}. This can be done in $O(n \log n)$ steps. The gap n^{c-1} between two different P_{i_1,j_1} and P_{i_2,j_2} is sufficient for the divide and conquer application for the disc covering problem in the last section since each disc radius is between 1 and another constant. We only design the algorithm a set of n grid points set P in an $n^c \times n^c$ region. It is not meaningful to consider the width $w \geq \sqrt{n}$ as our upper bound $w\sqrt{n}$ is even larger than the total number of points.

Definition 3. *Let P be a set of grid points on the plane. A $\frac{2}{3}$-boundary is a line L such that the number of points of P on one side of L is in the interval $(\frac{2}{3}|P|, \frac{2}{3}|P| + 1]$. For a $\frac{2}{3}$-boundary L, if L' is another $\frac{2}{3}$-boundary for P such that L and L' are parallel each other, and there are $\geq \frac{1}{3}|P|$ points between them, we call L and L' are a pair of $\frac{2}{3}$-boundaries. For a line L and vector v, if L can be expressed by the equation $p(t) = p_0 + t \cdot v$, then we say that the line L is along direction v. A set of vectors $v_1, v_2, \cdots, v_m$ is called a m-star vectors if the angle between v_i and v_{i+1} is $\frac{\pi}{m}$ for $i = 1, 2, \cdots, m-1$. If $L_1, L_2, \cdots, L_m$ are m lines through a same point and each L_i is along v_i, we call $L_1, L_2, \cdots, L_m$ m-star for the m-star vectors $v_1, v_2, \cdots, v_m$.*

It is easy to see that each $\frac{2}{3}$ center point is between every pair of $\frac{2}{3}$-boundaries. Assume that P is a set of n grid points in an $n^c \times n^c$ area S, where c is a constant. The function $f(L, S, P)$ computes the number of points of P on the two sides of the line L. For a vector v, if $p_i p_j$ is not parallel to v for any two points $p_i \neq p_j$ in P, it always exists a pair of $\frac{2}{3}$-boundaries along the direction v. If the angle between v and $p_i p_j$ is $> \frac{1}{n^{100}}$ for any $p_i \neq p_j$ in P, such a pair of boundaries can be found by binary search via checking the number of points of P on two sides of each line, which can be done by calling functin $f(L, S, P)$. It only checks $O(\log n)$ lines along the vector v. The idea of our algorithm is to find a m-star such that each line of the m-star is between a pair of $\frac{2}{3}$-boundaries. Therefore, each of them gives a balanced partition for the point set P. With high probability, each line also has angle $> \frac{1}{n^{100}}$ with any $p_i p_j$ for every $p_i \neq p_j$ in P. Select one of the m-lines L that has the least number of points from P to close L.

4.1 Intersection Between a Polygon and a Strip Area

We use a linked list to store the vertices of a convex polygon in counterclockwise order. A strip area is an area between two parallel lines on the plane. For two parallel lines L_1 and L_2 on the plane, we use $[L_1, L_2]$ to represent the strip region between L_1 and L_2. Each node of the linked list holds a vertex of the polygon. Throughout the algorithm, we often compute the intersection of a strip and a polygon. If the polygon has m nodes, such an intersection can be computed in $O(m)$ steps. For each line segment in the polygon, we check if there is a

intersection between it and the strip boundary lines. Record the area of the polygon inside the strip area.

4.2 Count the Number of Points on the Two Sides of a Line

Assume P is a set of n points in $n^c \times n^c$ square S_0. The square S_0 is partitioned into 4 squares S_1, S_2, S_3, S_4 of the same size. Each S_i is partitioned into smaller and smaller squares until the square size is less than 1×1. We obtain a tree of squares which has the largest square S_0 as root and all the squares in the same level have the same size. The depth of the tree is $O(\log n)$. The squares in this tree are called simple square. Each simple square S is assigned a counter denoted by $count(S)$, which counts the number of points in it.

Lemma 7. *For a set of grid points P of n points on the plane, there is an $O(n \log n)$-time algorithm to computer $count(S)$ for all of those simple squares S that contains at least one point.*

Proof. For each square S with at least one point from the set P, set up a counter for it. For each point p, start from the bottom-most square which contains $p \in P$, increase the counter by one for each simple square which contains p. Since each point only has $O(\log n)$ simple squares that contain it, it takes $O(n \log n)$ steps to set up those counters.

Algorithm
Input: a line L, a square S_0 of size $n^c \times n^c$, a set of n grid points P inside S_0.
Output: n_1 and n_2 that are the numbers of points of P on the left side and the right side of L, respectively.
$f(L, S_0, P)$
  $n_1 = n_2 = 0$;
  for the 4 sub-squares S_1, S_2, S_3, S_4 of S_0
    if $(S_i \cap L = \emptyset)$ then
      if S_i is on the left of L, then $n_1 = count(S_i) + n_1$.
      else $n_2 = count(S_i) + n_2$.
    let $S_{i_1}, \cdots, S_{i_k} (k \leq 4)$ be all squares from S_1, S_2, S_3, S_4 that
      $S_{i_j} \cap L \neq \emptyset$ and $count(S_{i_j}) > 0$ $(j = 1, \cdots, k)$.
    $(n_{i_j,1}, n_{i_j,2}) = f(L, S_{i_j})$ for $(j = 1, \cdots, k)$.
    $n_1 = n_1 + (n_{j_1,1} + \cdots + n_{j_k,1})$ and $n_2 = n_2 + (n_{j_1,2} + \cdots + n_{j_k,2})$
  return (n_1, n_2).
End of Algorithm

Lemma 8. *The running time for $f(L, S_0, P)$ is $O(t)$, where t is the number of simple squares $s \in S_0$ that touch L and have $count(s) > 0$.*

Proof. Going through the recursion, we only go to the next level of squares that touch the line L.

4.3 The Algorithm and Its Time Complexity

Definition 4. *For two lines L_1 and L_2,* share(L_1, L_2) *is the number of simple squares that intersect both L_1 and L_2.*

Let $\delta > 0$ be a small constant and $m = c_0\sqrt{n}$ for some constant $c_0 > 0$, which will be fixed at the end of the proof for Lemma 15. The algorithm below finds the separator for a set of n grid points in $n^c \times n^c$ region.

Algorithm

select a random 2D m-star vectors $v_1, v_2, \cdots, v_m$
find the pairs of $\frac{2}{3}$-boundaries $(L_{1,1}, L_{1,2})$ and $(L_{2,1}, L_{2,2})$ along
    the directions v_1 and v_2 respectively
let S be the intersection of two strips $[L_{1,1}, L_{1,2}]$ and $[L_{2,1}, L_{2,2}]$
for $(i = 3$ to $m)$ do
    find the pair of $\frac{2}{3}$-boundaries $(L_{i,1}, L_{i,2})$ along direction v_i
    let S be the intersection between S and the strip region $[L_{i,1}, L_{i,2}]$
$m_0 = \infty$
select a point $p \in S$
for $i = 1$ to m
    let L_i be a line through p
    if (measure$(L_i, P, a) < m_0$) then $m_0 =$ measure(L_i, P, a) and
        $L = L_i$
return L

End of Algorithm

Lemma 9. *During the first loop, S is an nonempty polygon all the time.*

Proof. The intersection between a convex polygon and a strip area is still convex polygon. By Lemma 6, S is nonempty all the time.

Lemma 10. *For two lines L_1 and L_2 with angle $0 < \theta \leq \frac{\pi}{2}$ between them, they share at most $\frac{c_1 \log n}{\sin\theta}$* simple squares *for some constant c_1.*

Proof. Let p be the intersection point of the two lines L_1 and L_2. If s_1 and s_2 are intersections between L_1, L_2 and a $t \times t$ square respectively, then dist$(s_1, s_2) \leq \sqrt{2}t$. It is easy to see that $dist(s_1, p) \leq \frac{\sqrt{2}t}{\sin\theta}$ and dist$(s_2, p) \leq \frac{\sqrt{2}t}{\sin\theta}$. Every point q in a $t \times t$ square that touches both L_1 and L_2 has distance $\leq \frac{\sqrt{2}t}{\sin\theta} + \sqrt{2}t$ to p. Furthermore, the point q has distance $\leq \sqrt{2}t$ to the middle line (through p) between L_1 and L_2 . Since those $t \times t$ squares do not overlap one other, the total number of them is $\leq \frac{2(\frac{\sqrt{2}t}{\sin\theta} + \sqrt{2}t)2\sqrt{2}t}{t^2} = 4\sqrt{2}(\frac{\sqrt{2}}{\sin\theta} + \sqrt{2}) \leq \frac{16}{\sin\theta}$. For some constant c_3, there are at most $c_3 \log n$ possible different sizes for the simple squares. Thus, L_1 and L_2 can share at most $\frac{16c_3 \log n}{\sin\theta}$ simple squares.

Lemma 11. *Let $v_1, v_2, \cdots, v_m$ be a m-star vectors. Each vector v_i has at most k lines along it (the line set along direction v_i is denoted by $L(v_i)$). Then for each line L_j in $L(v_j)$, $\sum_{i=1, i\neq j}^{m} \sum_{L_i \in L(v_i)}$* share$(L_j, L_i) \leq c_4 k \cdot m \cdot (\log m) \cdot (\log n)$ *for some constant $c_4 > 0$.*

Proof. For $L_i \in L(v_i)$, the angle between L_i and L_j is $\frac{\pi|i-j|}{m}$. By Lemma 10, $\text{share}(L_j, L_i) \leq \frac{c_1 \log n}{\sin \frac{|i-j|\pi}{m}} \leq \frac{c_2 m \log n}{\pi|i-j|}$ for some constant c_2. Therefore,

$$\sum_{i=1, i\neq j}^{m} \sum_{L_i \in L(v_i)} \text{share}(L_j, L_i) \leq \sum_{i=1, i\neq j}^{m} \sum_{L_i \in L(v_i)} \frac{c_2 m \log n}{\pi|i-j|} \leq \sum_{i=1, i\neq j}^{m} \frac{kc_2 m \log n}{\pi|i-j|}$$

$$\leq \frac{kc_2 m \log n}{\pi} \sum_{i=1, i\neq j}^{m} \frac{1}{|i-j|} < \frac{2kc_2 m \log n}{\pi} \sum_{i=1}^{m} \frac{1}{i} \leq c_4 \cdot k \cdot m \cdot (\log m) \cdot (\log n),$$

where c_4 is a constant $> \frac{2c_2}{\pi}$.

Lemma 12. *Let $\theta \leq \frac{\pi}{4m}$. Let $M_1, \cdots, M_t$ be t fixed line. Let $L_1, \cdots, L_m$ be the m lines along the m directions in a random m-star vectors $v_1, \cdots, v_m$, respectively. Then with probability $\leq \frac{4\theta \cdot m \cdot t}{\pi}$, one of $M_1, M_2, \cdots, M_t$ has angle $\leq \theta$ with some line from $L_1, \cdots, L_m$.*

Proof. We assume that the vector v_1 has an angle between 0 to $\frac{\pi}{m}$ with x-axis. Each M_j can have angle $\leq \theta$ with at most one line among $L_1, \cdots, L_m$. For a line L_i with angle to x-axis between $\frac{k\pi}{m}$ and $\frac{(k+1)\pi}{m}$, it has probability $\leq \frac{2\theta}{\frac{\pi}{m}} = \frac{2\theta m}{\pi}$ to have angle $\leq \theta$ with M_j. Therefore, the probability is $\leq \frac{4\theta m}{\pi} \cdot t$ to have one line $M_i \in \{M_1, \cdots, M_t\}$ such that that M_i has angle $\leq \theta$ with one of the vectors $L_1, L_2, \cdots, L_m$.

Lemma 13. *Let v be a vector and P be a set of n grid points in a $n^c \times n^c$. The vector v has angle $\geq \theta$ with any line $p_i p_j$ for every $p_i \neq p_j$ in P. It generate $O(\log n + \log \frac{1}{\theta})$ lines L along v (to query $f(L, S_0, P)$) to find out a pair of $\frac{2}{3}$ boundaries at direction v.*

Proof. We assume that v is along the direction of y-axis. For each point p on the plane, let $p(x)$ be the x-coordinate of p. Since the angle between y-axis and $p_i p_j$ is $\geq \theta$ and $\text{dist}(p_i, p_j) \geq 1$, we have $|p(x_i) - p(x_j)| \geq \sin\theta$. Let L_1 and L_2 be two vertical lines of distance $\leq n^c$ such that all points of P are between them. Let L be the middle vertical lines between L_1 and L_2. Let $(n_1, n_2) = f(L, S_0, P)$. If $n_1 < \frac{n}{3}$, then let $L_1 = L$. Otherwise, let $L_2 = L$. Repeat the binary search until one $\frac{2}{3}$-boundary line is found. After $O(\log n + \log \frac{1}{\theta})$ queries the function $f(L, S_0, P)$, the distance between two lines L_1 and L_2 is $< \sin\theta$.

Lemma 14. *Let $v_1, v_2, \cdots, v_m$ be a random m-star vectors. Let $h_0 > 2$ be a constant and $\theta = \frac{\pi}{4m^{h_0}}$. If for every two points $p_i, p_j \in P$, $p_i p_j$ has angle $\geq \theta$ with $v_k (k = 1, \cdots, m)$. Then the algorithm spends $O(n(\log n)^4)$ for finding the separator.*

Proof. In order to compute measure(L, P, a), we let L' and L'' be two lines on the left and right sides of L respectively, and both of them are parallel to L. Furthermore, both L' and L'' have distance a to L. Let $(n'_1, n'_2) = f(L', S_0, a)$ and $(n''_1, n''_2) = f(L'', S_0, a)$. Since all points of P with distance $\leq a$ to L are between L' and L'', measure$(L, P, a) = n - n'_1 - n''_2$.

Let $L(v_i)$ be the set of all lines L along v_i that are used to query the function $f(L, S_0, P)$ in the algorithm. The set $L(v_i)$ includes the lines (along v_i) for finding the the pair of $\frac{2}{3}$-boundaries along the v_i and also the line L_i' and L_i'' for computing measure(L_i, P, a). It is easy to see that the computational time of the algorithm is propositional to the number times that the lines in $\cup_{i=1}^{m} L(v_i)$ touch the simple squares s with $count(s) > 0$.

For a square s, assume s is touched by the lines in $U_1 \cup U_2 \cdots U_m$, where $U_i \subseteq L(v_i)$ $(i = 1, \cdots, m)$. If $U_1 = U_2 = \cdots = U_m = \emptyset$, s is called of type 0. If there exists only one i $(1 \leq i \leq m)$ with $U_i \neq \emptyset$, s is called of type 1. Otherwise, s is of type 2 (there exist $i \neq j$ with $U_i \neq \emptyset$ and $U_j \neq \emptyset$). For each v_i, $|L(v_i)| \leq c_5 \log n$ for some constant c_5. This is because that $L(v_i)$ is generated during the binary search for a pair of $\frac{2}{3}$-boundaries and the set $L(v_i)$ has $O(\log n)$ lines the along v_i (by Lemma 13 with $m = O(\sqrt{n})$ and $\theta = \frac{1}{m^{O(1)}}$). Define touch$(s)$ to be the number of lines in $\cup_{i=1}^{m} L(v_i)$ that intersects the simple square s.

There are only $O(n \log n)$ simple squares s that has points in P ($count(s) > 0$). Since $|L(v_i)| \leq c_5 \log n$, $\sum_{s \text{ is of type 1 and } count(s) > 0} \text{touch}(s)$ $= O(n(\log n)^2)$. For the set of of all type 2 simple squares,

$$\begin{aligned}
&\sum_{s \text{ is of type 2 and } count(s) > 0} \text{touch}(s) \\
&\leq 2 \sum_{j=1}^{m} \sum_{L_j \in L(v_j)} \Big(\sum_{i=1, i \neq j}^{m} \sum_{L_i \in L(v_i)} \text{share}(L_j, L_i) \Big) \\
&\leq 2 \sum_{j=1}^{m} \sum_{L_j \in L(v_j)} c_5 \cdot \log n \cdot c_4 \cdot m \cdot (\log m)(\log n) \text{(by Lemma 11 with } k \leq c_5 \log n) \\
&\leq 2 | \cup_{j=1}^{m} L(v_j)| \cdot c_5 \cdot \log n \cdot c_4 \cdot m \cdot (\log m)(\log n) \\
&\leq 2m \cdot (c_5 \log n) \cdot c_5 \cdot c_4 \cdot m \cdot (\log n)^3 = O(n \cdot (\log n)^4).
\end{aligned}$$

Combining the two cases above, we conclude that

$$\begin{aligned}
&\sum_{s \text{ is a simple square}} \text{touch}(s) \\
&= \sum_{s \text{ is of type 0}} \text{touch}(s) + \sum_{s \text{ is of type 1 and } count(s) > 0} \text{touch}(s) + \\
&\quad \sum_{s \text{ is of type 2 and } count(s) > 0} \text{touch}(s) \\
&= 0 + O(n \log n)^2 + O(n(\log n)^4) = O(n(\log n)^4).
\end{aligned}$$

Lemma 15. *Let $L_1, L_2, \cdots, L_m$ be a m-star through the same point o. There is a line L_i such that P has $\leq (\frac{4a}{\sqrt{\pi}}) \cdot \sqrt{n} + \delta\sqrt{n}$ grid points from P with distance $\leq a$ to L_i.*

Proof. For a grid point p, the number of lines that p has $\leq a$ distance to them is $\leq 2\arcsin\frac{a}{\mathrm{dist}(p,o)}\cdot\frac{m}{\pi}+1$. The total number of cases is $T=\sum_{i=1}^{n}(2\arcsin\frac{a}{\mathrm{dist}(p_i,o)}\cdot\frac{m}{\pi}+1)=\frac{2m}{\pi}\sum_{i=1}^{n}(\arcsin\frac{a}{\mathrm{dist}(p_i,o)})+n$. We present an upper bound for $\sum_{i=1}^{n}(\arcsin\frac{a}{\mathrm{dist}(p_i,o)})$ by using the method as [6].

Let $\epsilon>0$ be a small constant which will be determined later. Select r_0 to be large enough such that for every point p with $dist(o,p)\geq r_0$, $\arcsin\frac{a}{\mathrm{dist}(o,p)}<(1+\epsilon)\frac{a}{\mathrm{dist}(o,p)}$ and $\frac{1}{\mathrm{dist}(o,p')}<\frac{1+\epsilon}{\mathrm{dist}(o,p)}$ for every point p' with $\mathrm{dist}(p',p)\leq\frac{\sqrt{2}}{2}$. Let P_1 be the set of all points p in P such that $\mathrm{dist}(o,p)<r_0$. The number of grid points in P_1 is no more than $\pi(r_0+\frac{\sqrt{2}}{2})^2$. For each point $p\in P_1$, $\arcsin\frac{a}{\mathrm{dist}(o,p)}\leq\frac{\pi}{2}$. Let r be the minimum radius of a circle C with center at o and contains n grid points. Let $r'=r+\frac{\sqrt{2}}{2}$. The circle C' of radius r' contains all the 1×1 unit grid squares with center at points of P. Therefore, $\sum_{i=1}^{n}\arcsin\frac{a}{\mathrm{dist}(p_i,o)}=\sum_{p\in P_1}\arcsin\frac{a}{\mathrm{dist}(p,o)}+\sum_{p\in P-P_1}\arcsin\frac{a}{\mathrm{dist}(p,o)}\leq\sum_{p\in P_1}\frac{\pi}{2}+\sum_{p\in P-P_1}\arcsin\frac{a}{\mathrm{dist}(o,p)}<\frac{\pi^2}{2}(r_0+\frac{\sqrt{2}}{2})^2+\sum_{p\in P-P_1}\frac{(1+\epsilon)a}{\mathrm{dist}(o,p)}\leq\frac{\pi^2}{2}(r_0+\frac{\sqrt{2}}{2})^2+a(1+\epsilon)^2\int\int_{C'}\frac{1}{\mathrm{dist}(o,p)}d_xd_y=a(1+\epsilon)^2\int_0^{2\pi}\int_0^{r'}\frac{\rho}{\rho}d_\rho d_\theta+\frac{\pi^2}{2}(r_0+\frac{\sqrt{2}}{2})^2=2a\pi(1+\epsilon)^2r'+\frac{\pi^2}{2}(r_0+\frac{\sqrt{2}}{2})^2$.

It is easy to verify that $r\leq\frac{1}{\sqrt{\pi}}\sqrt{n}+4\sqrt{2}$ (see Lemma 9 in [6]). Therefore, there is a line L_i that has $\leq\frac{T}{m}\leq\frac{\frac{2m}{\pi}(2a\pi(1+\epsilon)^2r'+\frac{\pi^2}{2}(r_0+\frac{\sqrt{2}}{2})^2)+n}{m}\leq(\frac{4a}{\sqrt{\pi}})\cdot\sqrt{n}+\delta\sqrt{n}$ grid points from P with distance $\leq a$ if ϵ is selected small enough and c_0 is big enough.

Theorem 3. *For constant $a>0$ and small constant $\delta>0$, there is an $O(n(\log n)^4)$-time randomized algorithm for finding a-width separator for a set of n grid points set P in a $n^{O(1)}\times n^{O(1)}$ region such that each side has $\leq\frac{2}{3}|P|+1$ points of P, and the number of points with distance to the center line of the separator is $\leq(\frac{4a}{\sqrt{\pi}})\cdot\sqrt{n}+\delta\sqrt{n}$.*

Proof. Let $\theta=\frac{\pi}{4m^{h_0}}$ for constant $h_0>2$. By Lemma 12, it has probability $\geq 1-\frac{1}{m^{h_0-1}}$ that for every two points $p_i,p_j\in P$, the line p_ip_j has angle $\geq\theta$ with any v_k among the random m-star $v_1,\cdots,v_m$. By Lemma 14, the computational time is $O(n(\log n)^4)$. By Lemma 15, we can find a line L_i that satisfies the requirements of the theorem.

This theorem implies the corollary below by combining with corollary 2.

Corollary 3. *Let $\epsilon>0$ be a constant and P be a H_c' problem. There exists an $O(n(\log n)^5)$ time randomized approximation algorithm to output $Q\subseteq P$ with $s(Q)\geq(1-\epsilon)s(\mathrm{opt}(P))$.*

Acknowledgement

We would like to thank the anonymous referees from COCOON'05 for their helpful comments.

References

1. N. Alon, P. Seymour, and R. Thomas, Planar Separator, SIAM J. Discr. Math., 7(2), (1990) 184-193.
2. T. Asano, P. Brass, and S. Sasahara, Disc covering problem with application to digital halftoning, International Conference on Computational Science and Applications, Assisi, Italy, May 14-17, 2004, Proceedings, Part III. Lecture Notes in Computer Science 3045, Springer 2004, pp. 11-21.
3. T. Asano, T. Matsui, and T. Tokuyama, Optimal roundings of sequences and matrices, 3(7), Nordic Journal of Computing, 2000, pp. 241-256.
4. H.N. Djidjev, On the problem of partitioning planar graphs, SIAM Journal on Discrete Mathematics, 3(2) June, 1982, pp. 229-240.
5. B. Fu, Theory and Application of Width Bounded Geometric Separator, Electronic Colloquium on Computational Complexity 2005, TR05-13.
6. B. Fu and W. Wang, A $2^{O(n^{1-1/d}\log n)}$-time algorithm for d-dimensional protein folding in the HP-model, In proceedings of 31st International Colloquium on Automata, Languages and Programming, July 12-26, 2004, pp.630-644.
7. H. Gazit, An improved algorithm for separating a planar graph, manuscript, USC, 1986.
8. R. Graham, M. Grötschel, and L. Lovász, Handbook of combinatorics (volume I), MIT Press, 1996
9. R. J. Lipton and R. Tarjan, A separator theorem for planar graph, SIAM J. Appl. Math. 36(1979) 177-189.
10. G.L. Miller, S.-H. Teng, and S. A. Vavasis, An unified geometric approach to graph separators. In 32nd Annual Symposium on Foundation of Computer Science, IEEE 1991, pp. 538-547.
11. V. Ostromoukhov, Pseudo-random halftone screening for color and black & white printing, Proceedings of the 9th congress on advances in non-impact printing technologies, Yohohama, 1993, pp. 579-581.
12. V. Ostromoukhov and R. D. Hersch, Stochastic clustered-dot dithering, Journal of electronic imaging, 4(8), 1999, pp. 439-445.
13. J. Pach and P.K. Agarwal, Combinatorial Geometry, Wiley-Interscience Publication, 1995.
14. S. Sasahara and T. Asano, Adaptive cluster arrangement for cluster-dot halftoning using bubble packing method, Proceeding of 7th Japan joint workshop on algorithms and computation, Sendai, July 2003, pp. 87-93.
15. W. D. Smith and N. C. Wormald, Application of geometric separator theorems, FOCS 1998, 232-243.
16. D. A. Spielman and S. H. Teng, Disk packings and planar separators, 12th Annual ACM Symposium on Computational Geometry, 1996, pp.349-358.

Efficient Algorithms for Intensity Map Splitting Problems in Radiation Therapy*

Xiaodong Wu

Department of Electrical and Computer Engineering
Department of Radiation Oncology
The University of Iowa
Iowa City, IA 52242, USA
xiaodong-wu@uiowa.edu

Abstract. In this paper, we study several interesting *intensity map splitting (IMSp)* problems that arise in Intensity-Modulated Radiation Therapy (IMRT), a state-of-the-art radiation therapy technique for cancer treatments. In current clinical practice, a multi-leaf collimator (MLC) with a *maximum leaf spread* is used to deliver the prescribed intensity maps (IMs). However, the maximum leaf spread of an MLC may require to split a large intensity map into several abutting sub-IMs each being delivered separately, which results in prolonged treatment time. Few IM splitting techniques reported in the literature has addressed the issue of treatment delivery efficiency for large IMs. We develop a unified approach for solving the IMSp problems while minimizing the total beam-on time in various settings. Our basic idea is to formulate the IMSp problem as computing a k-link shortest path in a directed acyclic graph. We carefully characterize the intrinsic structures of the graph, yielding efficient algorithms for the IMSp problems.

Keywords: Intensity map splitting, k-link shortest paths, Algorithms, IMRT, Computational Medicine

1 Introduction

The *intensity map splitting* (IMSp) problems that we study in this paper arise in *Intensity-Modulated Radiation Therapy* (IMRT) [19], a state-of-the-art radiation therapy technique for cancer treatments. IMRT aims to deliver a highly conformal radiation dose to a target tumor while sparing the surrounding normal tissues. The quality of IMRT crucially depends on the ability to accurately and efficiently deliver the prescribed dose distributions of radiation, commonly called *intensity maps* (IMs). An intensity map is specified by a set of nonnegative integers on a 2-D grid (see Figure 1(a)). The number in a grid cell indicates the amount (in unit) of radiation to be delivered to the body region corresponding to that cell.

* This research was supported in part by a faculty start-up fund from the University of Iowa, Iowa City, IA 52242, USA. Part of this work was done at the Department of Computer Science, the University of Texas - Pan American, Edinburg, TX 78541.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 504–513, 2005.

One of the most advanced tools today for delivering IMs is the *multileaf collimator* (MLC) [19]. An MLC has multiple pairs of tungsten leaves of the same rectangular shape and size (see Figure 1(f)). The two opposite leaves of each pair are aligned to each other. The leaves can move left and right to form (say) an y-monotone rectilinear region (i.e., monotone to the y-axis), called an *MLC-aperture*. The cross-section of a cylindrical radiation beam (generated by a radiotherapy machine) is shaped by this MLC-aperture to deliver certain units of radiation to (a portion of) an IM. We associate that MLC-aperture with an integer representing the amount of radiation delivered by it. The mechanical constraints of the MLC limit what kinds of aperture shapes are allowed to be used [19]. One such constraint is called the *no-interleaf collision*: The distance between two opposite leaves of the neighboring pairs must be $\geq$ a given separation value δ (e.g., $\delta = 1cm$). For example, the Elekta MLC is subject to the no-interleaf collision constraint, while the Varian MLC allows interleaf collision. Another common constraint is called the *maximum leaf spread*: The two leaves of the same pair have an opening $\leq$ a given threshold Δ (e.g., $\Delta = 14.5cm$). Geometrically, the maximum leaf spread means the rectilinear y-monotone polygon corresponding to each MLC-aperture has a maximum horizontal "width" $\leq \Delta$.

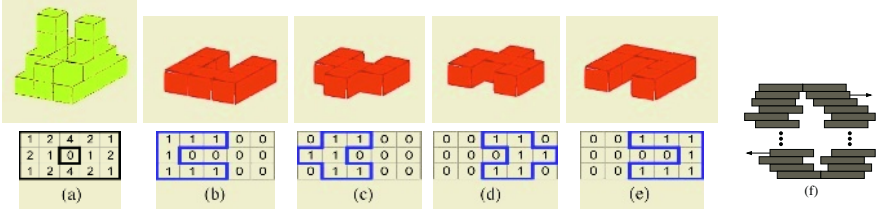


Fig. 1. (a) An intensity map A (bottom) and the corresponding 3-D IM mountain (top). (b) – (e) Four MLC-apertures S_k's (bottom) and the corresponding plateaus (top) of a unit height for building the 3-D IM mountain in (a). (f) A multileaf collimator.

One of the most popular IMRT approaches for delivering IMs using an MLC is the "step-and-shoot" technique [6, 19, 20]. Mathematically, the "step-and-shoot" delivery planning can be viewed as the following segmentation problem: Given an intensity map A defined on a 2-D $m \times n$ grid, decompose A into the form of $A = \sum_{k=1}^{\kappa} \alpha_k S_k$, where S_k is a special 0-1 matrix specifying an MLC-aperture, α_k is the amount of radiation delivered through S_k (called the *height* of S_k), and κ is the number of MLC-apertures used to deliver A (see Figure 1(a)-(e)). (The reader is referred to [2, 5, 8, 20] for more details on the step-and-shoot IMRT technique.) Intuitively, one may view the SLS problem as playing the following game: An IM is a 3-D "mountain" made of unit cubes (Figure 1(a)). Delivering an MLC-aperture can be viewed as creating a plateau whose height is the amount of radiation delivered. Then, one likes to "build" this mountain by stacking up the minimum number of plateaus with minimum total height (Figures 1(b)–(e)). It is important to note that each plateau, like the 3-D IMB mountain, is not one whole "rigid" object, i.e., it is made of unit cubes as well.

There are two major measures for the quality of the step-and-shoot delivery: (1) the *beam-on time* which is given by $\sum_{k=1}^{\kappa} \alpha_k$, and (2) the number κ of the MLC-apertures used in the delivery plan. The beam-on time is the actual time that the patient is exposed under the radiation beams. Minimizing beam-on time is crucial to reduce the patient's risk under irradiation and to reduce the delivery error caused by the tumor motion [2]. On the other hand, minimizing the number of MLC-apertures (hence, reducing the treatment time of each IM) is also important because it not only lowers the treatment cost for each patient but also enables hospitals to treat more patients [5].

In current clinical radiation therapy, large intensity maps frequently occur [7, 9, 16]. Due to the maximum leaf spread constraint of the MLC design, a large IM needs to be split into several abutting sub-IMs each being delivered separately using the step-and-shoot delivery technique. This results in prolonged treatment times (longer beam-on times and more MLC-apertures). Although the step-and-shoot delivery has received a great deal of attention from several research communities, such as medical physics [3, 6, 14, 15, 17, 18, 20], computer science [4, 5, 11, 12], and operations research [1, 2, 8], few IM splitting techniques reported in the literature has addressed the issue of treatment delivery efficiency for large IMs. To our best knowledge, Kamath *et al.* [13] first gave a quadratic time algorithm to split a large IM into *at most three* sub-IMs (thus restricting the maximum size of a large IM) while minimizing the total beam-on time.

In this paper, we study the following **intensity map splitting (IMSp)** problem: Given an $m \times n$ IM A and an integral maximum leaf spread $\Delta > 0$, split A into a set of abutting sub-IMs $\{M_1, M_2, \ldots, M_{\lceil \frac{n}{\Delta} \rceil}\}$ each A_i with size of $m \times n_i$ $(i = 1, 2, \ldots, \kappa = \lceil \frac{n}{\Delta} \rceil)$, such that: (1) $n_i \leq \Delta$, (2) $\sum_{i=1}^{\kappa} n_i = n$; and (3) the total beam-on time for delivering those sub-IMs is minimized. Note that $\kappa = \lceil \frac{n}{\Delta} \rceil$ is the minimum number of sub-IMs needed for delivering A subject to the maximum leaf spread constraint.

We develop a unified approach for solving the IMSp problem in various settings. In our solution, the IMSp problem is formulated as computing a k-link shortest path on a directed acyclic graph (DAG) transformed from the given IM. By judiciously characterizing the intrinsic structures, we compute the k-link shortest path without explicitly constructing the graph and integrate the computation of the beam-on times for sub-IMs into our k-link shortest path computation, which yields an improvement of the running time by at least an order of magnitude. Our main results in this paper are summarized as follows.

- An optimal $O(n)$ time algorithm for solving the IMSp problem with $m = 1$ (i.e., using only *one* MLC leaf pair for the IM delivery).
- An $O(mn\Delta)$ time algorithm for solving the IMSp problem in which the interleaf collisions are allowed.
- An $O(m^2 n\Delta)$ time algorithm for solving the IMSp problem subject to the no-interleaf collision constraint.

2 Reformulation of the IMSp Problem

Given an instance of the IMSp problem, an $m \times n$ IM A and an integer $\Delta > 0$ (the maximum leaf spread), we define a weighted directed acyclic graph $G = (V, E)$ for A, as follows.

The vertices of G are defined as $V = \{s, t\} \cup \{v_j : 0 \leq j < n\}$. Each column $A[*, j] = \{A[0, j], A[1, j], \ldots, A[m-1, j]\}$ $(j = 0, 1, \ldots, n-1)$ of A corresponds to exact one vertex v_j in G. For every $j \in \{0, 1, \ldots, n-2\}$, vertex v_j has a directed edge in E to each vertex in $\{v_k : j < k \leq \min\{j + \Delta, n-1\}\}$ (i.e., v_j connects to its following Δ vertices). Note that every edge $e = (v_j, v_{j'}) \in E$ $(j < j')$ is associated with a sub-IM of A, denoted by $A[*, j+1 \,..\, j']$, which consists of all rows of A from column $j+1$ to column j'. Let $T_{bot}(A')$ denote the minimum beam-on time for delivering the IM A'. The weight $w(e)$ of the edge $e = (v_j, v_{j'})$ is $T_{bot}(A[*, j+1 \,..\, j'])$, which can be computed using algorithms in [8, 10]. From vertex s, we introduce a directed edge $(s, v_j) \in E$ to every vertex v_j for $0 \leq j < \Delta$ and the weight of the edge equals to $T_{bot}(A[*, 0 \,..\, j])$. Meanwhile, each vertex v_j with $n - \Delta \leq j < n-1$ has a directed edge to vertex t whose weight is the minimum beam-on time for delivering the sub-IM $A[*, j+1 \,..\, n-1]$.

Obviously, $G = (V, E)$ thus constructed from A is a weighted directed acyclic graph (DAG). We next show that a κ-link shortest path ($\kappa = \lceil \frac{n}{\Delta} \rceil$) from s to t in G specifies an optimal splitting of A. Note that each κ-link s-to-t path p in G is of the form of $s \rightarrow v_{j_0} \rightarrow v_{j_1} \rightarrow \ldots \rightarrow v_{j_{\kappa-2}} \rightarrow t$; further, p defines a set of κ abutting sub-IMs $\mathcal{A} = \{A[*, 0 \,..\, j_0], A[*, j_0 + 1 \,..\, j_1], \ldots, A[*, j_{\kappa-2} + 1 \,..\, n-1]\}$ used for delivering the IM A. The total minimum beam-on time of $\mathcal{A}$ (i.e., $T_{bot}(\mathcal{A}) = \sum_{M \in \mathcal{A}} T_{bot}(M)$) equals to the total edge weight of p. Hence, we have the following lemma.

Lemma 1. *A κ-link shortest path from s to t in G specifies an optimal splitting $\mathcal{A}$ of the given IM A.*

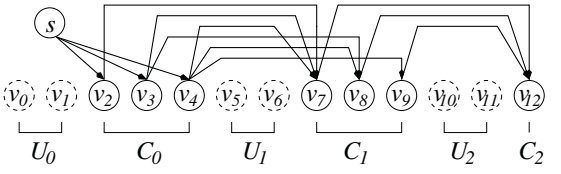
The DAG G has $O(n)$ vertices and $O(\Delta \cdot n)$ edges. Thus, it takes $O(\kappa \cdot \Delta \cdot n) = O(n^2)$ time to compute a κ-link shortest path in G. Let $T(G)$ denote the total time complexity for constructing G from A, which depends on the computation of the minimum beam-on time of an intensity map [8, 10]. Hence, the IMSp problem is solvable in $O(n^2 + T(G))$ time.

The k-link s-to-t path is a straightforward model for solving the IMSp problem. Next, we further exploit the intrinsic structures of the IMSp problem to simplify G and give a new dynamic programming approach.

Let $\mu = n \bmod \Delta$ and if $\mu = 0$, set $\mu = \Delta$. Then, we partition the column index set $\{0, 1, \ldots, n-1\}$ of A into the following consecutive segments: $C_k = \{j : k \cdot \Delta + \mu - 1 \leq j < (k+1) \cdot \Delta\}$ and $U_k = \{j : k \cdot \Delta \leq j < k \cdot \Delta + \mu - 1\}$ for $k = 0, 1, \ldots, \kappa - 2$; and $C_{\kappa-1} = \{n-1\}$ and $U_{\kappa-1} = \{j : (\kappa-1) \cdot \Delta \leq j < n-1\}$ (note that U_k's are empty when $\mu = 1$). Figure 2(b) illustrates the partition for the sample IM in Figure 2(a) with $\Delta = 5$. Let $\mathcal{A} = \{A[*, 0 \,..\, j_0], A[*, j_0 + 1 \,..\, j_1], \ldots, A[*, j_{\kappa-2} + 1 \,..\, n-1]\}$ be a feasible splitting of A, simply denoted by a set $\mathcal{A} = \{j_0, j_1, \ldots, j_{\kappa-1}\}$ with $j_{\kappa-1} = n-1$. The following lemmas establish

0	3	6	5	2	7	8	1	6	9	5	0	0
2	4	1	1	6	6	9	12	12	16	8	6	2
1	1	3	12	9	16	16	7	8	12	8	3	1

(a)



(b)

Fig. 2. Illustrating the construction of the graph G. (a) An example IM A with size of 3×13 and the maximum leaf spread $\Delta = 5$. (b) The graph G constructed from the IM A in (a).

the connection between a feasible splitting $\mathcal{A}$ of A and the segments $\{C_k : 0 \leq k < \kappa\}$.

Lemma 2. *For any feasible splitting* $\mathcal{A} = \{j_0, j_1, \ldots, j_{\kappa-1}\}$ *of* A, $\mathcal{A} \cap \left(\bigcup_{k=0}^{\kappa-1} U_k\right) = \varnothing$.

Proof. If U_k's are empty, obviously, $\mathcal{A} \cap \left(\bigcup_{k=0}^{\kappa-1} U_k\right) = \varnothing$. Otherwise, we prove this lemma by contradiction.

Assume that there exists a feasible splitting $\mathcal{A}$ and $q \in \mathcal{A}$ such that $q \in U_r$ $(0 \leq r < \kappa)$. Thus, $q \geq r \cdot \Delta$. Hence, the sub-IM $A[*, 0\,..\,q]$ needs to be split into at least $(r+1)$ sub-IMs subject to the maximum leaf spread constraint, while $A[*, q+1\,..\,n-1]$ needs at least $(\kappa - r)$ sub-IMs. Therefore, the total number of sub-IMs needed is at least $\kappa + 1$, which is a contradiction.

Based on Lemma 2, any $j \in \mathcal{A}$ is in $\bigcup_{k=0}^{\kappa-1} C_k$. Let us investigate the distribution of $\mathcal{A} = \{j_0, j_1, \ldots, j_{\kappa-1}\}$ in the segments C_k's. When $\kappa = 2$, note that $j_1 \in \mathcal{A}$ equals to $n-1$ (i.e., $j_1 \in C_1 = \{n-1\}$), it is then clear that $j_0 \in C_0$. Next, we consider $\kappa > 2$ and claim that for each $k = 0, 1, \ldots, \kappa-1$, $j_k \in C_k$. We assume otherwise, that is, for any $k = 0, 1, \ldots, \kappa-1$, $|C_k \cap \mathcal{A}|$ either equals to 0 or ≥ 2. Notice that if there exists C_r that $|C_r \cap \mathcal{A}| \geq 2$, then there must have an C_q such that $C_q \cap \mathcal{A} = \varnothing$ since $|\mathcal{A}| = \kappa$. Thus, there must exist an C_q with $|C_q \cap \mathcal{A}| = 0$. Then, there is a sub-IM $A[*, j'\,..\,j'']$ in $\mathcal{A}$ such that $j' < q \cdot \Delta + \mu - 1$ and $j'' \geq (q+1) \cdot \Delta$, which indicates that $j'' - j' > \Delta$. Hence, $\mathcal{A}$ is not a feasible splitting of A, a contradiction. Thus, the following lemma holds.

Lemma 3. *In any feasible splitting* $\mathcal{A} = \{j_0, j_1, \ldots, j_{\kappa-1}\}$ *of* A, $j_k \in C_k$ *for* $k = 0, 1, \ldots, \kappa-1$.

Lemma 3 leads to a simple dynamic programming approach for solving the IMSp problem, as follows. Let $S(v_j)$ denote the minimum total beam-on time for delivering the sub-IM $A[*, 0\,..\,j]$ subject to the maximum leaf spread constraint. Note that if $j \in C_k$ $(k = 1, 2, \ldots, \kappa-2)$, then any j' with $j - \Delta \leq j' < k \cdot \Delta$ is in C_{k-1}. Thus, we have

$$S(v_j) = \begin{cases} w(s, v_j) & \text{if } j \in C_0 \\ \min\{S(v_{j'}) + w(v_{j'+1}, v_j) : & \\ \quad j - \Delta \le j' < k \cdot \Delta\} & \text{if } j \in C_k \ (1 \le k < \kappa) \end{cases} \quad (1)$$

Hence, $S(v_{n-1})$ is the minimum total beam-on time for delivering A, which defines an optimal splitting $\mathcal{A}^*$. Based on Lemma 3, we can remove all the vertices v_j's with $j \in \bigcup_{i=0}^{\kappa-1} U_i$ from G, thus simplifying the construction of G. Figure 2(b) shows an example graph G thus constructed from the IM in Figure 2(a) with $\Delta = 5$. We still use G to denote the resulting graph. The running time of this dynamic programming scheme is clearly $O(\Delta \cdot n)$ in the worst case after constructing the graph G, which takes $T(G)$ time.

Lemma 4. *The IMSp problem can be solved in $O(\Delta \cdot n + T(G))$ time.*

Note that G has $O(\Delta \cdot n)$ edges each associated with one sub-IM of A, for which we need to compute the minimum beam-on time. This may become the bottleneck of our IMSp algorithm. In the following sections, we integrate the computation of the beam-on time [8, 10] into our dynamic programming scheme to further improve our algorithm for the IMSp solving problem in various setting.

3 The IMSp Problem with $m = 1$

This section presents our optimal $O(n)$ time algorithm for computing an optimal splitting of a given IM A with *only* one row of intensities (i.e., $m = 1$). For example, this case occurs when using one pair of MLC leaves for delivering the intensity map. Thus, we assume that the input IM A is a vector, i.e., $A = (A[0], A[1], \ldots, A[n-1])$. Kamath *et al.* [13] gave an $O(n^2)$ algorithm for this case while $n \le 3\Delta$. We consider in this section an arbitrary n and Δ.

The minimum beam-on time $T_{bot}(B)$ of an IM B ($m = 1$) can be computed [8], as follows:

$$T_{bot}(B) = B[0] + \sum_{j=2}^{n-1} \max\{0, B[j] - B[j-1]\}.$$

Let $\mathcal{A} = \{j_0, j_1, \ldots, j_{\kappa-1}\}$ with $j_{\kappa-1} = n-1$ be a feasible splitting of the given IM A. Then,

$$T_{bot}(\mathcal{A}) = \left(A[0] + \sum_{j=2}^{j_0} \max\{0, A[j] - A[j-1]\} \right) + \sum_{k=1}^{\kappa-1} \left(A[j_{k-1}+1] + \sum_{j=j_{k-1}+2}^{j_k} \max\{0, A[j] - A[j-1]\} \right)$$

Note that for any $0 < j < n$, $A[j] = \max\{0, A[j] - A[j-1]\} + \min\{A[j-1], A[j]\}$. Thus,

$$T_{bot}(\mathcal{A}) = T_{bot}(A) + \sum_{k=0}^{\kappa-2} \min\{A[j_k + 1], A[j_k]\}.$$

We call $\sum_{k=0}^{\kappa-2} \min\{A[j_k+1], A[j_k]\}$ the *bot-increase* of a feasible splitting $\mathcal{A}$ of A. Hence, the following lemma, which generalizes the result in [13], follows.

Lemma 5. *A feasible splitting $\mathcal{A} = \{j_0, j_1, \ldots, j_{\kappa-2}, n-1\}$ that minimizes the bot-increase among all possible feasible splitting of A gives an optimal solution to the IMSp problem.*

Based on Lemmas 3 and 5, the following dynamic programming scheme is used to compute an optimal splitting of A. Let $S(j)$ be the minimum *bot*-increase among all possible feasible splitting of $A[0\,..\,j]$. Then,

$$S(j) = \begin{cases} \min\{A[j], A[j-1]\} & \text{if } j \in C_0 \\ \min\{S(j') : j - \Delta \le j' < k \cdot \Delta\} & \\ \quad + \min\{A[j], A[j-1]\} & \text{if } j \in C_k \ (1 \le k < \kappa) \end{cases} \tag{2}$$

Thus, for each $j \in C_k$ $(1 \le k < \kappa)$, we need to compute the minimum of $S(j')$'s for $j - \Delta \le j' < k \cdot \Delta$. A straightforward way takes $O(\Delta)$ time and the total running time is $O(\Delta \cdot n)$. However, we can do better to achieve an optimal linear time algorithm. We use an additional array L of size $O(\Delta)$ to keep the minimum of $\{S(i), S(i+1), \ldots, S(k \cdot \Delta - 1)\}$ for each $i = j - \Delta, j - \Delta + 1, \ldots, k \cdot \Delta - 1$. It is easy to see that array L can be computed in $O(\Delta)$. Hence, for each $j \in C_k$ $(1 \le k < \kappa)$, $\min\{S(j') : j - \Delta \le j' < k \cdot \Delta\} + \min\{A[j], A[j-1]\}$ can be obtained in $O(1)$ time by using array L. Therefore, $S(n-1)$ can be computed in $O(n)$ time.

Theorem 1. *The IMSp problem with $m = 1$ is solvable in an optimal $O(n)$ time.*

4 The IMSp Problem Allowing the Interleaf Collisions

In this section, we give our $O(mn\Delta)$ time algorithm for solving the IMSp problem in which the interleaf collisions are allowed.

As shown in Section 2, the graph G constructed from the given IM A has $O(\Delta \cdot n)$ edges. For each edge in G, we need to compute the minimum bean-on time for its associated sub-IM of size $O(m \times \Delta)$. The minimum beam-on time $T_{bot}(B)$ of an $m' \times n'$ IM B can be computed as shown in [8]:

$$T_{bot}(B) = \max_{i \in \{0,1,\ldots,m'-1\}} \left\{ B[i,0] + \sum_{j=1}^{n'-1} \max\{0, B[i,j] - B[i,j-1]\} \right\},$$

which takes $O(m'n')$ time. Thus, the time $T(G)$ for constructing the graph G is $O(mn\Delta^2)$. Hence, our algorithm in Section 2 for solving the IMSp problem allowing the interleaf collisions runs in $O(mn\Delta^2)$ time.

However, by carefully characterizing the computation of the minimum beam-on time, we are able to improve the algorithm by at least $O(\Delta)$. The value of $(B[i,0] + \sum_{j=1}^{n-1} \max\{0, B[i,j] - B[i,j-1]\})$ is the minimum beam-on time of

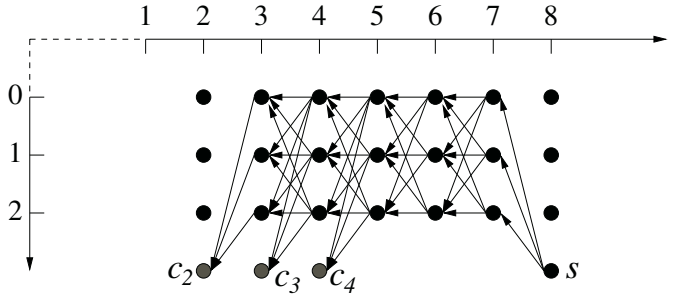


Fig. 3. Illustrating the construction of the graph H_q for computing $S(v_q)$ for the IM in Figure 2(a) with $\Delta = 5$ and $q = 7$.

the i-th row $B[i, *]$ of B. Thus, $T_{bot}(B) = \max_{i=0,1,\ldots,m'-1} T_{bot}(B[i, *])$. Then, consider a sub-IM $A[*, j' .. j]$ of A $(j > j' > 0)$. The minimum beam-on time $T_{bot}(A[*, j' .. j]) = \max_{i=0,1,\ldots,m-1} T_{bot}(A[i, j' .. j])$. Note that,

$$T_{bot}(A[i, j' .. j]) = T_{bot}(A[i, 0 .. j]) - T_{bot}(A[i, 0 .. j'-1]) + \min\{A[i, j'-1], A[i, j']\}.$$

Hence, for every row i of A, we compute $T_{bot}(i, 0 .. j)$ for each $j \in \{0, 1, \ldots, n-1\}$, which totally takes $O(n)$ time. Then, $T_{bot}(A[i, j' .. j])$ can be computed in $O(1)$ time. Thus, $S(v_{n-1})$, the minimum total beam-on time for delivering A subject to the maximum leaf spread constraint, can be computed in $O(mn\Delta)$ time based on the recursive Equation (1).

Theorem 2. *The IMSp problem allowing interleaf collisions is solvable in $O(mn\Delta)$ time for a given $m \times n$ IM A and the maximum leaf spread Δ.*

5 The IMSp Problem Subject to the No-interleaf Collision

This section presents our $O(m^2 n\Delta)$ time algorithm for solving the IMSp problem, in which an MLC subject to the no-interleaf collision constraint is used for delivering the given $m \times n$ IM A.

Recall that $S(v_q)$ denotes the minimum total beam-on time for delivering the sub-IM $A[*, 0 .. q]$ subject to the maximum leaf spread constraint. From Equation (1),

$$S(v_q) = \min\{S(v_j) + w(v_{j+1}, v_q) : q - \Delta \le j < k \cdot \Delta\},$$

if $q \in C_k$ $(k = 1, 2, \ldots, \kappa - 1)$. For a given $q \in C_k$, instead of computing $w(v_{j+1}, v_q)$ for each possible j (i.e., $q - \Delta \le j < k \cdot \Delta$) first, we compute all $w(v_{j+1}, v_q)$'s in one shot by formulating it as a single-source longest path problem in a weighted directed acyclic graph. Then, clearly $S(v_q)$ can be computed in $O(\Delta)$ time.

The DAG $H_q = (V_q, E_q)$ for computing all $w(v_{j+1}, v_q)$'s is constructed in the following way. The vertices of H_q are defined as $V_q = \{s\} \cup \{c_j : q - \Delta \le j <$

$k \cdot \Delta\} \cup \{u_{i,j} : 0 \le i < m, q - \Delta < j \le q\}$ (note that $q \in C_k$). The edge set E_q contains four different types of edges, E_s, E_b, and E_c, that is, $E_q = E_s \cup E_b \cup E_c$, where

$$\begin{aligned} E_s &= \{(s, u_{i,q}) : 0 \le i < m\}, \\ E_b &= \{(u_{i,j}, u_{i',j-1}) : 0 \le i, i' < m, q - \Delta + 1 < j \le q\}, \\ E_c &= \{(u_{i,j}, c_{j-1}) : 0 \le i < m, q - \Delta \le j < k \cdot \Delta\}. \end{aligned}$$

Next, we define the *cost* $w'(e)$ for each edge $e \in E_q$.

$$w'(e) = \begin{cases} A[i,q] & \text{if } e = (s, u_{i,q}) \in E_s, \\ \max\{0, A[i', j-1] - A[i', j]\} & \\ \quad - \sum_{p=i}^{i'-1} A[p,j] & \text{if } e = (u_{i,j}, u_{i',j-1}) \\ & \quad \in E_b \text{ and } i \le i', \\ \max\{0, A[i', j-1] - A[i', j]\} & \\ \quad - \sum_{p=i'+1}^{i} A[p,j] & \text{if } e = (u_{i,j}, u_{i',j-1}) \\ & \quad \in E_b \text{ and } i > i', \\ 0 & \text{if } e = (u_{i,j}, c_{j-1}) \in E_c. \end{cases} \tag{3}$$

Figure 3 shows an example graph H_q for the IM in Figure 2(a) with $q = 7$. Using similar techniques in [10], we are able to prove that the total cost of the longest path from vertex s to c_j $(q - \Delta \le j < k \cdot \Delta)$ equals to the minimum beam-on time of the sub-IM $A[*, j+1\,..\,q]$ (i.e., $w(v_{j+1}, v_q)$). Note that H_q is a DAG with $O(m \cdot \Delta)$ and $O(m^2 \cdot \Delta)$ edges. Thus, it takes $O(m^2 \cdot \Delta)$ time for computing a shortest s-to-t path in H_q. Note that in the worst case, we need to compute $S(v_q)$ for every $q = 0, 1, \ldots, n-1$. Hence, Theorem 3 follows.

Theorem 3. *The IMSp problem subject to the no-interleaf collision constraint can be solved in $O(m^2 n \Delta)$ time.*

6 Conclusion

In this paper, we study the intensity map splitting problems that seek to split a large intensity map into several deliverable sub-IMs while minimizing the total beam-on time. Our unified approach leads to efficient algorithms for solving the IMSp problems in various settings. When using only one pair of MLC leaves to deliver the IMs, our algorithm is optimal for the IMSp problem.

References

1. R.K. Ahuja and H.W. Hamacher, Minimizing Beam-on Time in Radiation Therapy Treatment Planning Using Network Flows, submitted to *Networks,* 2003.
2. N. Boland, H.W. Hamacher, and F. Lenzen, Minimizing Beam-on Time in Cancer Radiation Treatment Using Multileaf Collimators, *Networks,* 43(2)(2004), pp. 226-240.

3. T.R. Bortfeld, D.L. Kahler, T.J. Waldron, and A.L. Boyer, X-ray Field Compensation with Multileaf Collimators, *Int. J. Radiat. Oncol. Biol. Phys.,* 28(1994), pp. 723-730.
4. D.Z. Chen, X.S. Hu, S. Luan, X. Wu, and C.X. Yu, Optimal Terrain Construction Problems and Applications in Intensity-Modulated Radiation Therapy, *Lecture Notes in Computer Science, Springer-Verlag, Proc. 10th Annual European Symp. on Algorithms (ESA),* Vol. 2461, pp. 270-283, 2002.
5. D.Z. Chen, X.S. Hu, S. Luan, C. Wang, and X. Wu, Geometric Algorithms for Static Leaf Sequencing Problems in Radiation Therapy, *Proc. of 19th ACM Symp. on Computational Geometry (SoCG),* San Diego, CA, June 2003, pp. 88-97.
6. J. Dai and Y. Zhu, Minimizing the Number of Segments in a Delivery Sequence for Intensity-Modulated Radiation Therapy with Multileaf Collimator, *Med. Phys.,* 28(10)(2001), pp. 2113-2120.
7. N. Dogan, L.B. Leybovich, A. Sethi, and B. Emami, Automatic Feathering of Split Fields for Step-and-Shoot Intensity Modulated Radiation Therapy, *Phys. Med. Biol.,* 48(2003), pp. 1133-1140.
8. K. Engel, A New Algorithm for Optimal Multileaf Collimator Field Segmentation, *http://www.trinity.edu/aholder/HealthApp/oncology/papers,* March 2003.
9. L. Hong, A. Kaled, C. Chui, T. LoSasso, M. Hunt, S. Spirou, J. Yang, H. Amols, C. Ling, Z. Fuks, and S. Leibel, IMRT of Large Fields: Whole-Abdomen Irradiation, *Int. J. Radiat. Oncol. Biol. Phys.,* 54(2002), pp. 278-289.
10. T. Kalinowski, An algorithm for optimal multileaf collimator field segmentation with interleaf collision constraint, *http://www.trinity.edu/aholder/HealthApp/oncology/papers,* February 2003.
11. S. Kamath, S. Sahni, J. Li, J. Palta, and S. Ranka, Leaf sequencing algorithms for segmented multileaf collimation, *Phys. in Med. and Biol.,* 48(3)(2003), pp. 307-324.
12. S. Kamath, S. Sahni, J. Palta, S. Ranka, and J. Li, Optimal leaf sequencing with elimination of tongue-and-groove underdosage, *Phys. in Med. and Biol.,* 49(2004), N7-N19.
13. S. Kamath, S. Sahni, S. Ranka, J. Li, and J. Palta, Optimal Field Splitting for Large Intensity-Modulated Fields, to appear in *Medical Physics.*
14. L.D. Potter, S.X. Chang, T.J. Cullip, and A.C. Siochi, A Quality and Efficiency Analysis of the $IMFAST^{TM}$ Segmentation Algorithm in Head and Neck "Step & Shoot" IMRT Treatments, *Med. Phys.,* 29(3)(2002), pp. 275-283.
15. W. Que, Comparison of Algorithms for Multileaf Collimator Field Segmentation, *Med. Phys.,* 26(1999), pp. 2390-2396.
16. Q. Wu, M. Arnfield, S. Tong, Y. Wu, and R. Mohan, Dynamic Splitting of Large Intensity-Modulated Fields, *Phys. Med. Biol.,* 45(2000), pp. 1731-1740.
17. R.A.C. Siochi, Minimizing Static Intensity Modulation Delivery Time Using an Intensity Solid Paradigm, *Int. J. Radiat. Oncol. Giol. Phys.,* 43(1999), pp. 671-680.
18. R. Svensson, P. Kallman, and A. Brahme, An Analytical Solution for the Dynamic Control of Multileaf Collimation, *Phys. in Med. and Biol.,* 39(1994), pp. 37-61.
19. S. Webb, *Intensity-Modulated Radiation Therapy*, Institute of Cancer Research and Royal Marsden NHS Trust, Jan. 2001.
20. P. Xia and L.J. Verhey, MLC Leaf Sequencing Algorithm for Intensity Modulated Beams with Multiple Static Segments, *Med. Phys.,* 25(1998), pp. 1424-1449.

Distributions of Points in d Dimensions and Large k-Point Simplices

Extended Abstract

Hanno Lefmann

Fakultät für Informatik, TU Chemnitz, D-09107 Chemnitz, Germany
lefmann@informatik.tu-chemnitz.de

Abstract. We consider a variant of Heilbronn's triangle problem by asking for fixed dimension $d \geq 2$ and for fixed integers $k \geq 3$ with $k \leq d+1$ for a distribution of n points in the d-dimensional unit-cube $[0,1]^d$ such that the minimum volume of a k-point simplex among these n points is as large as possible. Denoting by $\Delta_{k,d}(n)$ the supremum of the minimum volume of a k-point simplex among n points over all distributions of n points in $[0,1]^d$ we will show that $c_k \cdot (\log n)^{1/(d-k+2)}/n^{(k-1)/(d-k+2)} \leq \Delta_{k,d}(n) \leq c'_k/n^{(k-1)/d}$ for $3 \leq k \leq d+1$, and moreover $\Delta_{k,d}(n) \leq c''_k/n^{(k-1)/d+(k-2)/(2d(d-1))}$ for $k \geq 4$ even, and constants $c_k, c'_k, c''_k > 0$.

1 Introduction

For integers $n \geq 3$, Heilbronn's problem asks for the supremum $\Delta_2(n)$ of the minimum area of a triangle formed by three of n points over all distributions of n points in the unit-square $[0,1]^2$. For primes n, the points $P_k = 1/n \cdot (k \bmod n, k^2 \bmod n)$, $k = 0, 1, \ldots, n-1$, show that $\Delta_2(n) = \Omega(1/n^2)$. Komlós, Pintz and Szemerédi [10] improved this to $\Delta_2(n) = \Omega(\log n/n^2)$, see [5] for a deterministic polynomial time algorithm achieving this lower bound on $\Delta_2(n)$, which is currently the best known. Upper bounds were proved in a series of papers by Roth [15–18] and Schmidt [19]. The currently best known upper bound is due to Komlós, Pintz and Szemerédi [9], who proved $\Delta_2(n) = O(2^{c\sqrt{\log n}}/n^{8/7})$ for some constant $c > 0$. We remark that for n points chosen uniformly at random and independently of each other from $[0,1]^2$, the expected value of the minimum area of a triangle among these n points is $\Theta(1/n^3)$, as was shown by Jiang, Li and Vitany [8].

A variant of Heilbronn's problem considered by Barequet asks, given a fixed dimension $d \geq 2$, for the supremum $\Delta_{d+1,d}(n)$ of the minimum volume of a $(d+1)$-point simplex among n points in the d-dimensional unit-cube $[0,1]^d$ over all distributions of n points in $[0,1]^d$. He showed in [2] the lower bound $\Delta_{d+1,d}(n) = \Omega(1/n^d)$, which was improved in [11] to $\Delta_{d+1,d}(n) = \Omega(\log n/n^d)$. In [14], a deterministic polynomial time algorithm was given achieving this lower bound on $\Delta_{4,3}(n)$. Recently, Brass [6] improved the known upper bound $\Delta_{d+1,d}(n) = O(1/n)$ to $\Delta_{d+1,d}(n) = O(1/n^{(2d+1)/(2d)})$ for odd $d \geq 3$. Moreover, an on-line version of this variant was investigated in [3] for dimensions $d = 3, 4$.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 514–523, 2005.

Here we consider the following generalization of Heilbronn's problem: given fixed integers d, k with $3 \le k \le d+1$, find for any integer $n \ge k$ a distribution of n points in the d-dimensional unit-cube $[0,1]^d$ such that the minimum volume of a k-point simplex among these n points is as large as possible. Let $\Delta_{k,d}(n)$ denote the corresponding supremum values – over all distributions of n points in $[0,1]^d$ – on the minimum volume of a k-point simplex among n points in $[0,1]^d$.

The parameter $\Delta_{3,d}(n)$, i.e. areas of triangles in $[0,1]^d$, was investigated by this author in [12], where it was shown that $c_3 \cdot (\log n)^{1/(d-1)}/n^{2/(d-1)} \le \Delta_{3,d}(n) \le c_3'/n^{2/d}$ for constants $c_3, c_3' > 0$. Here we prove the following bounds.

Theorem 1. *Let d, k be fixed integers with $3 \le k \le d+1$. Then, for constants $c_k, c_k', c_k'' > 0$, which depend on k, d only, for every integer $n \ge k$ it is*

$$c_k \cdot \frac{(\log n)^{1/(d-k+2)}}{n^{(k-1)/(d-k+2)}} \le \Delta_{k,d}(n) \le \frac{c_k'}{n^{(k-1)/d}} \qquad \textit{for } k \textit{ odd} \tag{1}$$

$$c_k \cdot \frac{(\log n)^{1/(d-k+2)}}{n^{(k-1)/(d-k+2)}} \le \Delta_{k,d}(n) \le \frac{c_k''}{n^{(k-1)/d+(k-2)/(2d(d-1))}} \qquad \textit{for } k \textit{ even.} \tag{2}$$

For $d = 2$ and $k = 3$, the lower bound (1) is just the result from [10]. For $k = d+1$, this yields the bounds from [6] and [11]. Indeed, our arguments for proving Theorem 1 yield a randomized polynomial time algorithm, which finds a distribution of n points in $[0,1]^d$ achieving these lower bounds.

2 A Lower Bound on $\Delta_{k,d}(n)$

Fist we introduce some notation which is used throughout this paper.

Let dist (P_i, P_j) be the *Euclidean distance* between the points P_i and P_j. A *simplex* given by k points $P_1, \ldots, P_k \in [0,1]^d$ is the set of all points $P_1 + \sum_{i=2}^{k} \lambda_i \cdot (P_i - P_1)$ with $\lambda_i \ge 0$, $i = 2, \ldots, k$, and $\sum_{i=2}^{k} \lambda_i \le 1$. The $(k-1)$-dimensional *volume of a k-point simplex* determined by the points $P_1, \ldots, P_k \in [0,1]^d$, $2 \le k \le d+1$, is defined by vol $(P_1, \ldots, P_k) := 1/(k-1)! \cdot \prod_{j=2}^{k}$ dist $(P_j; < P_1, \ldots, P_{j-1} >)$, where dist $(P_j; < P_1, \ldots, P_{j-1} >)$ denotes the Euclidean distance of the point P_j from the affine space $< P_1, \ldots, P_{j-1} >$ generated by $P_1, \ldots, P_{j-1}$ with $< P_1 > := P_1$.

In our arguments we will use hypergraphs. A hypergraph $\mathcal{G} = (V, \mathcal{E})$ with vertex set V and edge set $\mathcal{E}$ is *k-uniform* if $|E| = k$ for all edges $E \in \mathcal{E}$. A subset $I \subseteq V$ of the vertex set V is *independent* if I contains no edges from $\mathcal{E}$. The largest size $|I|$ of an independent set in $\mathcal{G}$ is the *independence number* $\alpha(\mathcal{G})$. A hypergraph $\mathcal{G} = (V, \mathcal{E})$ is *linear* if $|E \cap E'| \le 1$ for all distinct edges $E, E' \in \mathcal{E}$.

First we prove the lower bound in (1), (2) from Theorem 1, namely that

$$\Delta_{k,d}(n) \ge c_k \cdot (\log n)^{1/(d-k+2)}/n^{(k-1)/(d-k+2)} \ . \tag{3}$$

Proof. Let d, k be fixed integers with $3 \le k \le d+1$. For arbitrary integers $n \ge k$ and a suitable constant $\alpha > 0$, we select uniformly at random and independently of each other $N := n^{1+\alpha}$ points $P_1, P_2, \ldots, P_N$ from $[0,1]^d$.

For certain values $D_j := N^{-\gamma_j}$ for some constants $\gamma_j > 0$, $j = 2, \ldots, k-1$, and some value $V_0 > 0$, where all these will be fixed later, we form a random hypergraph $\mathcal{G} = \mathcal{G}(D_2, \ldots, D_{k-1}, V_0) = (V, \mathcal{E}_2 \cup \cdots \cup \mathcal{E}_k)$ with vertex set $V = \{1, 2, \ldots, N\}$, where vertex i corresponds to the random point $P_i \in [0,1]^d$, and with j-element edges, $j = 2, \ldots, k$. For $j = 2, \ldots, k-1$, let $\{i_1, \ldots, i_j\} \in \mathcal{E}_j$ be a j-element edge if and only if vol $(P_{i_1}, \ldots, P_{i_j}) \leq D_j$. Moreover, let $\{i_1, \ldots, i_k\} \in \mathcal{E}_k$ be a k-element edge if and only if vol $(P_{i_1}, \ldots, P_{i_k}) \leq V_0$ and $\{i_1, \ldots, i_k\}$ does not contain any j-element edges $E \in \mathcal{E}_j$ for $j = 2, \ldots, k-1$. An independent set $I \subseteq V$ in this hypergraph $\mathcal{G}$ yields $|I|$ many points in $[0,1]^d$ such that each k-point simplex among these $|I|$ points has volume bigger than V_0. Our aim is to show the existence of a large independent set $I \subseteq V$ in $\mathcal{G}$. For doing so, we will use a result on the independence number of linear k-uniform hypergraphs due to Ajtai, Komlós, Pintz, Spencer and Szemerédi [1], see [7].

Theorem 2. [1, 7] *Let $k \geq 3$ be a fixed integer. Let $\mathcal{G} = (V, \mathcal{E})$ be a k-uniform hypergraph on $|V| = n$ vertices with average degree $t^{k-1} = k \cdot |\mathcal{E}|/|V|$. If $\mathcal{G}$ is linear, then for some constant $c_k^* > 0$ its independence number $\alpha(\mathcal{G})$ satisfies*

$$\alpha(\mathcal{G}) \geq c_k^* \cdot \frac{n}{t} \cdot \log^{\frac{1}{k-1}} t \, . \tag{4}$$

The difficulty in our arguments is, to find a certain subhypergraph of our random non-uniform hypergraph $\mathcal{G}$ to which we can apply Theorem 2. For doing so, we will select a random induced subhypergraph $\mathcal{G}^*$ of $\mathcal{G}$ by controling certain parameters of $\mathcal{G}^*$. For $j = 2, \ldots, k-1$, let $|BP_j(\mathcal{G})|$ be a random variable counting the number of 'bad j-pairs of simplices' in $\mathcal{G}$, which are among the N random points $P_1, \ldots, P_N \in [0,1]^d$ those unordered pairs of k-point simplices arising from $\mathcal{E}_k$, which share j vertices. We will show that in the random nonuniform hypergraph $\mathcal{G}$ the expected numbers $E(|\mathcal{E}_i|)$ and $E(|BP_j(\mathcal{G})|)$ of i-element edges and of 'bad j-pairs of simplices' arising from $\mathcal{E}_k$, respectively, $i, j = 2, \ldots, k-1$, are not too big. Then in a certain induced subhypergraph of $\mathcal{G}$, which will be obtained by a random selection of vertices from V, we will delete one vertex from each i-element edge $E \in \mathcal{E}_i$ and from each 'bad j-pair of simplices' arising from $\mathcal{E}_k$, $i, j = 2, \ldots, k-1$. This yields a *k-uniform linear* subhypergraph $\mathcal{G}^* = (V^*, \mathcal{E}_k^*)$ of $\mathcal{G}$, thus $\mathcal{G}^*$ fulfills the assumptions of Theorem 2 and then we can apply it.

Lemma 1. *For $i = 2, \ldots, k$ with $2 \leq k \leq d+1$ and random points $P_1, \ldots, P_i \in [0,1]^d$ for constants $c_i^* > 0$ and a real $V > 0$ it is*

$$Prob\ (vol\ (P_1, \ldots, P_i) \leq V) \leq c_i^* \cdot V^{d-i+2} \, . \tag{5}$$

Proof. Let $P_1, \ldots, P_i$ be i random points in $[0,1]^d$. We may assume that the i points are numbered in such a way that for $2 \leq g \leq h \leq i$ it is

$$\text{dist } (P_g; < P_1, \ldots, P_{g-1} >) \geq \text{dist } (P_h; < P_1, \ldots, P_{g-1} >) \, . \tag{6}$$

The point P_1 can be anywhere in $[0,1]^d$. Given the point P_1, the probability, that its Euclidean distance from the point $P_2 \in [0,1]^d$ is within the infinitesimal

range $[r_1, r_1 + dr_1]$, is at most the difference of the volumes of the d-dimensional balls with center P_1 and with radii $(r_1 + dr_1)$ and r_1, respectively, hence

$$\text{Prob}\ (r_1 \leq \text{ dist}\ (P_1, P_2) \leq r_1 + dr_1) \leq d \cdot C_d \cdot r_1^{d-1} dr_1\ ,$$

where throughout this paper C_d denotes the value of the volume of the d-dimensional unit-ball in $\mathbb{R}^d$ with $C_1 := 2$.

Given the points P_1 and P_2 with dist $(P_1, P_2) = r_1$, the probability that the distance dist $(P_3; < P_1, P_2 >)$ of the point $P_3 \in [0,1]^d$ from the line $< P_1, P_2 >$ is within the infinitesimal range $[r_2, r_2 + dr_2]$ is at most the difference of the volumes of cylinders centered at the line $< P_1, P_2 >$ with radii $r_2 + dr_2$ and r_2, respectively, and, by assumption (6), with height $2 \cdot r_1 = 2 \cdot \text{dist}\ (P_1, P_2)$, thus

$$\text{Prob}\ (r_2 \leq \text{ dist}\ (P_3; < P_1, P_2 >) \leq r_2 + dr_2) \leq 2 \cdot r_1 \cdot (d-1) \cdot C_{d-1} \cdot r_2^{d-2} dr_2.$$

In general, by condition (6), given the points $P_1, \ldots, P_g$, $g < i$, with dist $(P_f; < P_1, \ldots, P_{f-1} >) = r_{f-1}$ for $f = 2, \ldots, g$, the projection of the point P_{g+1} onto the affine space $< P_1, \ldots, P_f >$ must lie in a shape of volume at most $2^{f-1} \cdot r_1 \cdot \ldots \cdot r_{f-1}$. Hence for $g < i - 1$ we obtain

$$\begin{aligned} &\text{Prob}\ (r_g \leq \text{dist}\ (P_{g+1}; < P_1, \ldots, P_g >) \leq r_g + dr_g) \\ &\leq 2^{g-1} \cdot r_1 \cdot \ldots \cdot r_{g-1} \cdot (d - g + 1) \cdot C_{d-g+1} \cdot r_g^{d-g} dr_g\ . \end{aligned}$$

For $g = i - 1$, however, to satisfy vol $(P_1, \ldots, P_i) \leq V$, we must have $1/(i-1)! \cdot \prod_{g=2}^{i} \text{dist}\ (P_g; < P_1, \ldots, P_{g-1} >) \leq V$, hence the projection of the point P_i onto the affine space $< P_1, \ldots, P_{i-1} >$ must lie in a shape of volume $2^{i-2} \cdot r_1 \cdot \ldots \cdot r_{i-2}$ and the point P_i has Euclidean distance at most $\frac{(i-1)! \cdot V}{r_1 \cdot \ldots \cdot r_{i-2}}$ from $< P_1, \ldots, P_{i-1} >$, which happens with probability at most

$$2^{i-2} \cdot r_1 \cdot \ldots \cdot r_{i-2} \cdot C_{d-i+2} \cdot \left(\frac{(i-1)! \cdot V}{r_1 \cdot \ldots \cdot r_{i-2}} \right)^{d-i+2}\ .$$

Thus for some constants $c_i^*, c_i^{**} > 0$ we infer

$$\begin{aligned} &\text{Prob}\ (\text{vol}\ (P_1, \ldots, P_i) \leq V) \\ &\leq \int_{r_{i-2}=0}^{\sqrt{d}} \ldots \int_{r_1=0}^{\sqrt{d}} 2^{i-2} \cdot C_{d-i+2} \cdot \frac{((i-1)! \cdot V)^{d-i+2}}{(r_1 \cdot \ldots \cdot r_{i-2})^{d-i+1}} \cdot \\ &\quad \cdot \prod_{g=1}^{i-2} \left(2^{g-1} \cdot r_1 \cdot \ldots \cdot r_{g-1} \cdot (d - g + 1) \cdot C_{d-g+1} \cdot r_g^{d-g} \right) dr_{i-2} \ldots dr_1 \leq \\ &\leq c_i^{**} \cdot V^{d-i+2} \cdot \int_{r_{i-2}=0}^{\sqrt{d}} \ldots \int_{r_1=0}^{\sqrt{d}} \prod_{g=1}^{i-2} r_g^{2i-2g-3} dr_{i-2} \ldots dr_1 \\ &\leq c_i^* \cdot V^{d-i+2} \qquad \text{as } 2 \cdot i - 2 \cdot g - 3 > 0. \end{aligned}$$

□

Corollary 1. *For $i = 2, \ldots, k-1$ with $2 \leq k \leq d+1$ and constants $c_i', c_k' > 0$, it is*

$$E(|\mathcal{E}_i|) \leq c_i' \cdot N^{i - \gamma_i(d-i+2)} \qquad \textit{and} \qquad E(|\mathcal{E}_k|) \leq c_k' \cdot V_0^{d-k+2} \cdot N^k\ . \tag{7}$$

Proof. There are $\binom{N}{i}$ possibilities to choose i out of the N random points $P_1, \ldots, P_N \in [0,1]^d$, and by (5) from Lemma 1 with $V := N^{-\gamma_i}$ for $i = 2, \ldots, k-1$ and $V := V_0$ for $i = k$ the inequalities (7) follow. □

Lemma 2. *For $j = 2, \ldots, k-1$ with $3 \le k \le d+1$ and constants $c'_{2,j} > 0$ it is*

$$E(|BP_j(\mathcal{G})|) \le c'_{2,j} \cdot V_0^{2(d-k+2)} \cdot N^{2k-j+\gamma_j(d-k+2)} \,. \tag{8}$$

Proof. For $j = 2, \ldots, k-1$, we show the upper bound $O(V_0^{2(d-k+2)} \cdot N^{\gamma_j(d-k+2)})$ on the probability that $2k-j$ random points, chosen uniformly and independently of each other in $[0,1]^d$, yield a 'bad j-pair of simplices'. Since there are $\binom{N}{2k-j}$ possibilities to choose $2k-j$ out of the N random points $P_1, \ldots, P_N \in [0,1]^d$, the upper bound (8) follows. There are $\binom{2k-j}{k}$ choices for k out of $2k-j$ points and $\binom{k}{j}$ possibilities to choose the j common points, say the two simplices are determined by the points $P_1, \ldots, P_k$ and $P_1, \ldots, P_j, Q_{j+1}, \ldots, Q_k$. By Lemma 1 we know that Prob (vol $(P_1, \ldots, P_k) \le V_0) \le c_k^* \cdot V_0^{d-k+2}$. If $\{P_1, \ldots, P_k\} \in \mathcal{E}_k$, then by construction of our hypergraph $\mathcal{G}$ we have vol $(P_1, \ldots, P_j) > N^{-\gamma_j}$, and we condition on this in the following. Given the points $P_1, \ldots, P_j, Q_{j+1}, \ldots, Q_g$, $g = j, \ldots, k-1$, with dist $(Q_f; < P_1, \ldots, P_j, Q_{j+1}, \ldots, Q_{f-1} >) = r_f$, $f = j+1, \ldots, g$, we infer for $g \le k-2$:

$$\begin{aligned} &\text{Prob } (r_g \le \text{dist } (Q_{g+1}; < P_1, \ldots, P_j, Q_{j+1}, \ldots, Q_g >) \le r_g + dr_g) \\ &\le (\sqrt{d})^{g-1} \cdot (d+1-g) \cdot C_{d+1-g} \cdot r_g^{d-g} dr_g \,, \end{aligned}$$

since all points Q_{g+1}, which satisfy dist $(Q_{g+1}; < P_1, \ldots, P_j, Q_{j+1}, \ldots, Q_g >) \le r$, are contained in a product of a $(g-1)$-dimensional shape of volume at most $(\sqrt{d})^{g-1}$ and a $(d+1-g)$-dimensional ball of radius r.

For $g = k-1$, having fixed the points $P_1, \ldots, P_j, Q_{j+1}, \ldots, Q_{k-1} \in [0,1]^d$, to fulfill vol $(P_1, \ldots, P_j, Q_{j+1}, \ldots, Q_k) \le V_0$, we must have

$$\frac{(j-1)!}{(k-1)!} \cdot \text{dist } (Q_k; < P_1, \ldots, P_j, Q_{j+1}, \ldots, Q_{k-1} >) \cdot \text{vol } (P_1, \ldots, P_j) \cdot \prod_{g=j}^{k-2} r_g \le V_0 \,,$$

and, using vol $(P_1, \ldots, P_j) > N^{-\gamma_j}$, this happens with probability at most

$$(\sqrt{d})^{k-2} \cdot C_{d-k+2} \cdot \left(\frac{(k-1)!}{(j-1)!} \cdot \frac{V_0 \cdot N^{\gamma_j}}{\prod_{g=j}^{k-2} r_g} \right)^{d-k+2} \,.$$

Putting all these probabilities together, we obtain for constants $c^*_{2,j}, c^{**}_{2,j} > 0$ the following upper bound, which finishes the proof of Lemma 2:

$$\begin{aligned}
&\text{Prob}\left(\{P_1,\dots,P_k\},\{P_1,\dots,P_j,Q_{j+1},\dots,Q_k\}\text{ is a 'bad } j\text{-pair of simplices'}\right)\\
&\le c_k^*\cdot V_0^{d-k+2}\cdot\int_{r_{k-2}=0}^{\sqrt{d}}\dots\int_{r_j=0}^{\sqrt{d}} d^{\frac{k-2}{2}}\cdot C_{d+2-k}\cdot\frac{(k-1)!^{d-k+2}}{(j-1)!^{d-k+2}}\cdot\\
&\cdot\left(\frac{V_0\cdot N^{\gamma_j}}{\prod_{g=j}^{k-2} r_g}\right)^{d-k+2}\cdot\prod_{g=j}^{k-2}\left(d^{\frac{g-1}{2}}\cdot(d+1-g)\cdot C_{d+1-g}\cdot r_g^{d-g}\right)dr_{k-2}\dots dr_j\le\\
&\le c_{2,j}^{**}\cdot V_0^{2(d-k+2)}\cdot N^{\gamma_j(d-k+2)}\cdot\int_{r_{k-2}=0}^{\sqrt{d}}\dots\int_{r_j=0}^{\sqrt{d}}\prod_{g=j}^{k-2} r_g^{k-g-2}dr_{k-2}\dots dr_j\\
&\le c_{2,j}^{*}\cdot V_0^{2(d-k+2)}\cdot N^{\gamma_j(d-k+2)}\qquad\text{as } k-g-2\ge 0.\qquad\square
\end{aligned}$$

Using (7) and (8) and Markov's inequality, there exist $N=n^{1+\alpha}$ points in the unit-cube $[0,1]^d$ such that the corresponding hypergraph $\mathcal{G}=(V,\mathcal{E}_2\cup\dots\cup\mathcal{E}_k)$ on $|V|=N$ vertices satisfies for $i,j=2,\dots,k-1$ and $3\le k\le d+1$:

$$|\mathcal{E}_i|\le 2k\cdot c_i'\cdot N^{i-\gamma_i(d-i+2)d} \tag{9}$$

$$|\mathcal{E}_k|\le 2k\cdot c_k'\cdot V_0^{d-k+2}\cdot N^k \tag{10}$$

$$|BP_j(\mathcal{G})|\le 2k\cdot c_{2,j}'\cdot V_0^{2(d-k+2)}\cdot N^{2k-j+\gamma_j(d-k+2)}\,. \tag{11}$$

By (10) the average degree $t^{k-1}:=k\cdot|\mathcal{E}_k|/|V|$ of $\mathcal{G}=(V,\mathcal{E}_2\cup\dots\cup\mathcal{E}_k)$ among the edges from $\mathcal{E}_k$ satisfies $t^{k-1}\le 2k^2\cdot c_k'\cdot V_0^{d-k+2}\cdot N^{k-1}=:t_0^{k-1}$. For some suitable constant $\varepsilon>0$, we pick uniformly at random and independently of each other vertices from V with probability $p:=N^\varepsilon/t_0\le 1$. Let $V^*\subseteq V$ be the random set of the chosen vertices, and let $\mathcal{G}^*=(V^*,\mathcal{E}_2^*\cup\dots\cup\mathcal{E}_k^*)$ with $\mathcal{E}_i^*:=\mathcal{E}_i\cap[V^*]^i$, $i=2,\dots,k$, be the resulting random induced subhypergraph of $\mathcal{G}$. By (9) – (11) we infer for the expected numbers of vertices, i-element edges and 'bad j-pairs of simplices' in $\mathcal{G}^*$, $i,j=2,\dots,k-1$, for constants $c_1,c_i,c_{2,j},c_k>0$:

$$\begin{aligned}
E(|V^*|)&=p\cdot N\ge c_1\cdot N^\varepsilon/V_0^{\frac{d-k+2}{k-1}}\\
E(|\mathcal{E}_i^*|)&=p^i\cdot|\mathcal{E}_i|\le p^i\cdot 2k\cdot c_i'\cdot N^{i-\gamma_i(d-i+2)}\le c_i\cdot N^{i\varepsilon-\gamma_i(d-i+2)}/V_0^{\frac{i(d-k+2)}{k-1}}\\
E(|\mathcal{E}_k^*|)&=p^k\cdot|\mathcal{E}_k|\le p^k\cdot 2k\cdot c_k'\cdot V_0^{d-k+2}\cdot N^k\le c_k\cdot N^{k\varepsilon}/V_0^{\frac{d-k+2}{k-1}}\\
E(|BP_j(\mathcal{G}^*)|)&=p^{2k-j}\cdot|BP_j(\mathcal{G})|\le c_{2,j}\cdot V_0^{\frac{(j-2)(d-k+2)}{k-1}}\cdot N^{(2k-j)\varepsilon+\gamma_j(d-k+2)}\,.
\end{aligned}$$

By Chernoff's and Markov's inequality there exists an induced subhypergraph $\mathcal{G}^*=(V^*,\mathcal{E}_2^*\cup\dots\cup\mathcal{E}_k^*)$ of $\mathcal{G}$, such that for $i,j=2,\dots,k-1$:

$$|V^*|\ge(c_1-o(1))\cdot N^\varepsilon/V_0^{\frac{d-k+2}{k-1}} \tag{12}$$

$$|\mathcal{E}_i^*|\le 2k\cdot c_i\cdot N^{i\varepsilon-\gamma_i(d-i+2)}/V_0^{\frac{i(d-k+2)}{k-1}} \tag{13}$$

$$|\mathcal{E}_k^*|\le 2k\cdot c_k\cdot N^{k\varepsilon}/V_0^{\frac{d-k+2}{k-1}} \tag{14}$$

$$|BP_j(\mathcal{G}^*)|\le 2k\cdot c_{2,j}\cdot V_0^{\frac{(j-2)(d-k+2)}{k-1}}\cdot N^{(2k-j)\varepsilon+\gamma_j(d-k+2)}\,. \tag{15}$$

Now we set for some suitable constant $c^* > 0$:

$$V_0 := c^* \cdot (\log n)^{\frac{1}{d-k+2}} / n^{\frac{k-1}{d-k+2}} \ . \tag{16}$$

Lemma 3. *For $j = 2, \ldots, k-1$ and for fixed $0 < \varepsilon < (j-1)/((2k-j-1)\cdot(1+\alpha)) - \gamma_j \cdot (d-k+2)/(2k-j-1)$ it is $|BP_j(\mathcal{G}^*)| = o(|V^*|)$.*

Proof. Using (12), (15) and (16) with $N = n^{1+\alpha}$, where $\alpha, \gamma_j > 0$ are constants, $j = 2, \ldots, k-1$, we have

$$\begin{aligned}
&|BP_j(\mathcal{G}^*)| = o(|V^*|) \\
\Longleftarrow\ & V_0^{\frac{(j-2)(d-k+2)}{k-1}} \cdot N^{(2k-j)\varepsilon+\gamma_j(d-k+2)} = o(N^\varepsilon / V_0^{\frac{d-k+2}{k-1}}) \\
\Longleftrightarrow\ & V_0^{\frac{(j-1)(d-k+2)}{k-1}} \cdot N^{(2k-j-1)\varepsilon+\gamma_j(d-k+2)} = o(1) \\
\Longleftrightarrow\ & n^{(1+\alpha)((2k-j-1)\varepsilon+\gamma_j(d-k+2))-(j-1)} \cdot \log^{\frac{j-1}{k-1}} n = o(1) \\
\Longleftarrow\ & \varepsilon < \frac{j-1}{(2k-j-1)\cdot(1+\alpha)} - \frac{\gamma_j \cdot (d-k+2)}{2k-j-1} \ .
\end{aligned}$$

□

Lemma 4. *For $i = 2, \ldots, k-1$ and fixed $0 < \varepsilon \le \gamma_i \cdot (d-i+2)/(i-1) - 1/(1+\alpha)$ it is $|\mathcal{E}_i^*| = o(|V^*|)$.*

Proof. By (12), (13) and (16), using $N = n^{1+\alpha}$, we infer

$$\begin{aligned}
&|\mathcal{E}_i^*| = o(|V^*|) \\
\Longleftarrow\ & N^{i\varepsilon-\gamma_i(d-i+2)} / V_0^{\frac{i(d-k+2)}{k-1}} = o(N^\varepsilon / V_0^{\frac{d-k+2}{k-1}}) \\
\Longleftrightarrow\ & N^{(i-1)\varepsilon-\gamma_i(d-i+2)} / V_0^{\frac{(i-1)(d-k+2)}{k-1}} = o(1) \\
\Longleftrightarrow\ & n^{(1+\alpha)((i-1)\varepsilon-\gamma_i(d-i+2))+(i-1)} / \log^{\frac{i-1}{k-1}} n = o(1) \\
\Longleftarrow\ & \varepsilon \le \frac{\gamma_i \cdot (d-i+2)}{i-1} - \frac{1}{1+\alpha} \ .
\end{aligned}$$

□

The assumptions in Lemmas 3 and 4 are satisfied for $\gamma_j := (j-1)/((d-k+5/2)(1+\alpha))$, $j = 2, \ldots, k-1$, and $\varepsilon := 1/(4kd(1+\alpha))$ and $\alpha := 1/(4kd)$, also $p = N^\varepsilon / t_0 \le 1$ holds. In the induced subhypergraph $\mathcal{G}^* = (V^*, \mathcal{E}_2^* \cup \cdots \cup \mathcal{E}_k^*)$ we delete one vertex from each i-element edge and from each 'bad j-pair of simplices', $i, j = 2, \ldots, k-1$. Let $V^{**} \subseteq V^*$ be the set of remaining vertices. The on V^{**} induced subhypergraph $\mathcal{G}^{**}$ of $\mathcal{G}^*$ is k-uniform, hence $\mathcal{G}^{**} = (V^{**}, \mathcal{E}_k^{**})$ with $\mathcal{E}_k^{**} := [V^{**}]^k \cap \mathcal{E}_k^*$, and fulfills $|V^{**}| = (1 - o(1)) \cdot |V^*|$ by Lemmas 3 and 4. By (12) and (14) we have $|V^{**}| \ge c_1/2 \cdot N^\varepsilon / V_0^{(d-k+2)/(k-1)}$ and $|\mathcal{E}_k^{**}| \le |\mathcal{E}_k^*| \le 2k \cdot c_k \cdot N^{k\varepsilon} / V_0^{(d-k+2)/(k-1)}$, hence $\mathcal{G}^{**}$ has average degree $t^{k-1} = k \cdot |\mathcal{E}_k^{**}| / |V^{**}| \le (4k^2 \cdot c_k / c_1) \cdot N^{(k-1)\varepsilon} =: t_1^{k-1}$. Now the assumptions of Theorem 2 are fulfilled by the k-uniform subhypergraph $\mathcal{G}^{**}$ of $\mathcal{G}$, as it is linear, and with (4) we obtain for constants $c_k^*, c', c_1, c_k, c^* > 0$:

$$\begin{aligned}
\alpha(\mathcal{G}) \geq \alpha(\mathcal{G}^{**}) &\geq c_k^* \cdot \frac{|V^{**}|}{t} \cdot \log^{1/(k-1)} t \geq c_k^* \cdot \frac{|V^{**}|}{t_1} \cdot \log^{1/(k-1)} t_1 \geq \\
&\geq c_k^* \cdot \frac{c_1^{k/(k-1)} \cdot N^{\varepsilon} / V_0^{(d-k+2)/(k-1)}}{2 \cdot (4k^2 \cdot c_k)^{1/(k-1)} \cdot N^{\varepsilon}} \cdot \left(\log \left(\frac{4k^2 \cdot c_k}{c_1} \cdot N^{(k-1)\varepsilon} \right)^{\frac{1}{k-1}} \right)^{\frac{1}{k-1}} \\
&\geq c' \cdot \log^{1/(k-1)} n / V_0^{(d-k+2)/(k-1)} \qquad \text{as } N = n^{1+\alpha} \\
&\geq c' \cdot (1/c^*)^{(d-k+2)/(k-1)} \cdot \log^{1/(k-1)} n \cdot \frac{n}{\log^{1/(k-1)} n} \geq n \,,
\end{aligned}$$

where the last inequality follows by choosing in (16) a sufficiently small constant $c^* > 0$. Thus the hypergraph $\mathcal{G}$ contains an independent set $I \subseteq V$ with $|I| = n$. These n vertices yield n points in $[0,1]^d$, such that each k-point simplex arising from these points has volume bigger than V_0, i.e. $\Delta_{k,d}(n) = \Omega((\log n)^{1/(d-k+2)} / n^{(k-1)/(d-k+2)})$, which finishes the proof of (3). □

3 An Upper Bound on $\Delta_{k,d}(n)$

Here we show the upper bounds in Theorem 1, namely that for fixed $2 \leq k \leq d+1$ and constants $c'_k, c''_k > 0$ it is $\Delta_{k,d}(n) \leq c'_k / n^{(k-1)/d}$, moreover $\Delta_{k,d}(n) \leq c''_k / n^{(k-1)/d + (k-2)/(2d(d-1))}$ for k even.

Proof. We prove first that $\Delta_{k,d}(n) \leq c'_k / n^{(k-1)/d}$ for some constant $c'_k > 0$ and $2 \leq k \leq d+1$. Given any n points $P_1, P_2, \ldots, P_n \in [0,1]^d$, for some value $D > 0$ we construct a graph $G = G(D) = (V, E)$ with vertex set $V = \{1, 2, \ldots, n\}$, where vertex i corresponds to the point $P_i \in [0,1]^d$, and edge set E with $\{i, j\} \in E$ being an edge if and only if dist $(P_i, P_j) \leq D$. An independent set $I \subseteq V$ in this graph $G = G(D)$ yields a subset $I' \subseteq \{P_1, P_2, \ldots, P_n\}$ of points in $[0,1]^d$ with Euclidean distance between any two distinct points bigger than D. Each ball $B_r(P)$ with center $P \in [0,1]^d$ and radius $r \leq 1$ satisfies vol $(B_r(P) \cap [0,1]^d) \geq$ vol $(B_r(P))/2^d$. The balls with radius $D/2$ and centers from an independent set I' have pairwise empty intersection. As each ball $B_{D/2}(P)$ has volume $C_d \cdot (D/2)^d$, we infer $|I'| \cdot C_d \cdot (D/2)^d / 2^d \leq$ vol $([0,1]^d) = 1$, and hence the independence number $\alpha(G)$ of G satisfies

$$\alpha(G) \leq \frac{4^d}{C_d \cdot D^d} \,. \tag{17}$$

For $D := c/n^{1/d}$ with $c := (2 \cdot (k-1) \cdot 4^d / C_d)^{1/d}$ a constant, the average degree t of $G(D)$ satisfies $t \geq 1$ for $n \geq 2^{d+1}$, hence by Turán's theorem, $\alpha(G) \geq n/(2 \cdot t)$. With (17) this yields

$$\frac{4^d}{C_d \cdot D^d} \geq \alpha(G) \geq \frac{n}{2 \cdot t} \quad \Longrightarrow \quad t \geq \frac{C_d}{2 \cdot 4^d} \cdot n \cdot D^d \geq k - 1 \,. \tag{18}$$

Hence there exists a vertex $i_1 \in V$ and $k-1$ edges $\{i_1, i_2\}, \ldots, \{i_1, i_k\} \in E$ incident at vertex i_1. By construction, each point $P_{i_j} \in [0,1]^d$, $j = 2, \ldots, k$,

satisfies dist $(P_{i_1}, P_{i_j}) \le D$, thus dist $(P_{i_j}; < P_{i_1}, P_{i_2}, \ldots, P_{i_{j-1}} >) \le c/n^{1/d}$ for $j = 2, \ldots, k$, which implies vol $(P_{i_1}, \ldots, P_{i_k}) \le (1/(k-1)!) \cdot c^{k-1}/n^{(k-1)/d}$, i.e. $\Delta_{k,d}(n) = O(1/n^{(k-1)/d})$.

For even $k \ge 4$ we are able to prove a better upper bound. From (18) we obtain $|E| = n \cdot t/2 \ge C_d \cdot n^2 \cdot D^d/4^{d+1}$. Now let $c := (d \cdot 4^{d+1}/C_{d-1})^{1/d}$ and $D := 1/n^{1/d}$. We adapt an argument of Brass [6]. Each edge $\{i,j\} \in E$ determines a direction (P_iP_j), which can be viewed as a vector of length 1. The minimum angular distance between these directions is at most

$$\left(\frac{d \cdot C_d}{C_{d-1} \cdot |E|}\right)^{1/(d-1)} \le \left(\frac{d \cdot 4^{d+1}}{C_{d-1} \cdot c^d \cdot n}\right)^{1/(d-1)} \le \frac{1}{n^{1/(d-1)}} \, .$$

Thus for some constant $c(d) > 0$ there exist $\binom{k}{2}$ directions (P_iP_j), $\{i,j\} \in E$, with pairwise angular distance at most $\phi := c(d)/n^{1/(d-1)}$. The corresponding set $E^* \subseteq E$ of edges covers a subset $S \subseteq V$ of at least k vertices G. Consider a minimum subset $E^{**} \subseteq E^*$ of edges, which covers a subset $S^* \subseteq S$ of exactly k vertices. This set E^{**} contains only independent edges and stars. We pick one vertex from each independent edge $E \in E^{**}$ and the center of each star. Let $S^{**} \subseteq S^*$ be the set of chosen vertices with $|S^{**}| = s \le k/2$.

For each vertex $v \in S^* \setminus S^{**}$ there exists an edge $\{v,w\} \in E^{**}$ for some vertex $w \in S^{**}$, hence dist $(P_v, P_w) \le D$. Thus for each vertex $u \in S^* \setminus (S^{**} \cup \{v\})$ there is some vertex $t \in S^{**} \cup \{w\}$ such that the angular distance between the directions (P_uP_t) and (P_wP_v) is at most ϕ. Thus, the Euclidean distance between the point P_u and the affine space generated by the points P_r, $r \in S^{**} \cup \{v\}$, is at most D. With $D = c/n^{1/d}$ and $\sin\phi \le \phi$ for $\phi \ge 0$, and $(s-1)! \cdot$ vol $(S^{**}) \le (\sqrt{d})^{s-1}$ we obtain for the volume of the simplex determined by the k points P_s, $s \in S^*$, the following upper bound, which finishes the proof of Theorem 1:

$$\text{vol } (P_{s^*}; s^* \in S^*) \le \frac{1}{(k-1)!} \cdot (\sqrt{d})^{s-1} \cdot D \cdot (D \cdot \sin\phi)^{k-s-1} \le$$

$$\le \frac{1}{(k-1)!} \cdot d^{(k-2)/4} \cdot D \cdot (D \cdot c(d)/n^{1/(d-1)})^{k/2-1} = \frac{d^{(k-2)/4} \cdot c(k)^{k/2-1}}{(k-1)! \cdot n^{\frac{k-1}{d} + \frac{k-2}{2d(d-1)}}} \, . \, \square$$

4 Concluding Remarks

Our arguments together with an algorithmic version of Theorem 2, see [4], yield a randomized polynomial time algorithm for obtaining a distribution of n points in $[0,1]^d$, which shows $\Delta_{k,d}(n) = \Omega((\log n)^{1/(k-1)}/n^{(k-1)/(d-k+2)})$ for fixed $3 \le k \le d+1$. It might be of interest to have a deterministic polynomial time algorithm achieving this lower bound, as well as investigating the case $k > d+1$, compare [13] for the case of dimension $d = 2$.

References

1. M. Ajtai, J. Komlós, J. Pintz, J. Spencer and E. Szemerédi, *Extremal Uncrowded Hypergraphs*, Journal of Combinatorial Theory Ser. A, 32, 1982, 321–335.

2. G. Barequet, *A Lower Bound for Heilbronn's Triangle Problem in d Dimensions,* SIAM Journal on Discrete Mathematics 14, 2001, 230–236.
3. G. Barequet, *The On-Line Heilbronn's Triangle Problem in Three and Four Dimensions,* Proc. 8rd Annual International Computing and Combinatorics Conference COCOON'2002, LNCS 2387, Springer, 2002, 360–369.
4. C. Bertram–Kretzberg and H. Lefmann, *The Algorithmic Aspects of Uncrowded Hypergraphs,* SIAM Journal on Computing 29, 1999, 201–230.
5. C. Bertram-Kretzberg, T. Hofmeister and H. Lefmann, *An Algorithm for Heilbronn's Problem,* SIAM Journal on Computing 30, 2000, 383–390.
6. P. Brass, *An Upper Bound for the d-Dimensional Heilbronn Triangle Problem,* SIAM Journal on Discrete Mathenmatics, to appear.
7. R. A. Duke, H. Lefmann and V. Rödl, *On Uncrowded Hypergraphs,* Random Structures & Algorithms 6, 1995, 209–212.
8. T. Jiang, M. Li and P. Vitany, *The Average Case Area of Heilbronn-type Triangles,* Random Structures & Algorithms 20, 2002, 206–219.
9. J. Komlós, J. Pintz and E. Szemerédi, *On Heilbronn's Triangle Problem,* Journal of the London Mathematical Society, 24, 1981, 385–396.
10. J. Komlós, J. Pintz and E. Szemerédi, *A Lower Bound for Heilbronn's Problem,* Journal of the London Mathematical Society, 25, 1982, 13–24.
11. H. Lefmann, *On Heilbronn's Problem in Higher Dimension*, Combinatorica 23, 2003, 669–680.
12. H. Lefmann, *Large Triangles in the d-Dimensional Unit-Cube*, Proc. 10th Annual International Conference Computing and Combinatorics COCOON'2004, eds. K.-Y. Chwa and J. I. Munro, LNCS 3106, Springer, 2004, 43–52.
13. H. Lefmann, *Distributions of Points in the Unit-Square and Large k-Gons*, Proc. 16th Symposium on Discrete Algorithms SODA'2005, ACM and SIAM, 241–250.
14. H. Lefmann and N. Schmitt, *A Deterministic Polynomial Time Algorithm for Heilbronn's Problem in Three Dimensions,* SIAM Journal on Computing 31, 2002, 1926–1947.
15. K. F. Roth, *On a Problem of Heilbronn,* Journal of the London Mathematical Society 26, 1951, 198–204.
16. K. F. Roth, *On a Problem of Heilbronn, II and III,* Proc. of the London Mathematical Society (3), 25, 1972, 193–212 and 543–549.
17. K. F. Roth, *Estimation of the Area of the Smallest Triangle Obtained by Selecting Three out of n Points in a Disc of Unit Area,* Proc. of Symposia in Pure Mathematics, 24, 1973, AMS, Providence, 251–262.
18. K. F. Roth, *Developments in Heilbronn's Triangle Problem,* Advances in Mathematics, 22, 1976, 364–385.
19. W. M. Schmidt, *On a Problem of Heilbronn,* Journal of the London Mathematical Society (2), 4, 1972, 545–550.

Exploring Simple Grid Polygons

Christian Icking[1], Tom Kamphans[2], Rolf Klein[2], and Elmar Langetepe[2]

[1] University of Hagen, Praktische Informatik VI, 58084 Hagen, Germany
[2] University of Bonn, Computer Science I, Römerstraße 164, 53117 Bonn, Germany

Abstract. We investigate the online exploration problem of a short-sighted mobile robot moving in an unknown cellular room without obstacles. The robot has a very limited sensor; it can determine only which of the four cells adjacent to its current position are free and which are blocked, i. e., unaccessible for the robot. Therefore, the robot must enter a cell in order to explore it. The robot has to visit each cell and to return to the start. Our interest is in a short exploration tour, i. e., in keeping the number of multiple cell visits small. For abitrary environments without holes we provide a strategy producing tours of length $S \leq C + \frac{1}{2}E - 3$, where C denotes the number of cells – the area – , and E denotes the number of boundary edges – the perimeter – of the given environment. Further, we show that our strategy is competitive with a factor of $\frac{4}{3}$, and give a lower bound of $\frac{7}{6}$ for our problem. This leaves a gap of only $\frac{1}{6}$ between the lower and the upper bound.

Keywords: Robot navigation, exploration, covering, online algorithms, competitive analysis, lower bounds, grid polygons

1 Introduction

Exploring an unknown environment and searching for a target in unknown position are among the basic tasks of autonomous mobile robots. Both problems have received a lot of attention in computational geometry and in robotics; see e. g. [3, 5, 9, 12, 13, 17].

We use a simple model for the robot and its environment: the robot is short-sighted, and the surrounding is subdivided by a rectangular integer grid, similar to a chessboard. Essentially, there are two motivations for using this model instead of a robot with a full vision system: First, even a laser scanner has a reliable range of only a few meters. Hence, the robot has to move towards more distant areas in order to explore them. Second, service robots like lawn mowers or cleaners need to get close to their work areas. The robot's sensors provide the information, which of the four neighbors of the currently occupied cell do not belong to the polygon and which ones do. The robot can enter the latter cells. The robot's task is to visit every cell inside the polygon and to return to the start cell. Sometimes, this task in also called *covering*.

Even though our robot does not know its environment in advance it is interesting to ask how short a tour can be in the offline situation, i. e., when the environment is already known. This amounts to constructing a shortest traveling salesperson tour on the free cells.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 524–533, 2005.

If the polygonal environment contains obstacles, the problem of finding such a minimum length tour is known to be NP-hard, see Itai et al. [14]. There are $1+\varepsilon$ approximation schemes by Grigni et al. [7], Arora [2], and Mitchell [16], and a $\frac{53}{40}$ approximation by Arkin et al. [1].

In a polygon without obstacles, the complexity of constructing offline a minimum length tour seems to be open. Ntafos [18] and Arkin et al. [1] have shown how to approximate the minimum length tour with factors of $\frac{4}{3}$ and $\frac{6}{5}$, respectively. Umans and Lenhart [19] have provided an $O(C^4)$ algorithm for deciding if there exists a Hamiltonian cycle, i. e., a tour that visits each of the C cells of a polygon *exactly* once. For the related problem of Hamiltonian paths, Everett [4] has given a polynomial algorithm for certain grid graphs.

In this paper our interest is in the online version of the cell exploration problem. Exploring a grid polygon *with* holes was considered by Icking et al. [10, 11] and independently by Gabriely and Rimon [6]. Icking et al. showed a lower bound of 2 for this problem and introduced an exploration strategy that needs no more than $C+\frac{1}{2}E+3H+W-2$ steps[1], see [15], where C denotes the number of cells, E the number of boundary edges, H the number of holes and W is a measure for the windings of the polygon. Gabriely and Rimon showed an upper bound of $C+B$, where B denotes the number of boundary cells.

We consider the exploration of polygons *without* holes. Although both problems seem to be closely related there is an important difference: We have a lower bound of 2 for polygons with holes, but it turns out that we can do much better in simple polygons.

An upper bound for our exploration strategy is given in terms of the polygon's *area*, C, and the *perimeter*, E. While C is the number of free cells, E is the number of edges between a free cell and a blocked cell, see for example Fig. 1. We use E to distinguish between skinny and thick environments. For thick environments, $E \in O(\sqrt{C})$ holds; thus, the number of additional cell visits is substantially smaller than C. Only in polygons that do not contain any 2×2-square of free cells, E achieves its maximum value of $2(C+1)$, and our upper bound is equal to $2C-2$, but in this case one cannot do better, since even the optimal offline strategy needs that number of steps.

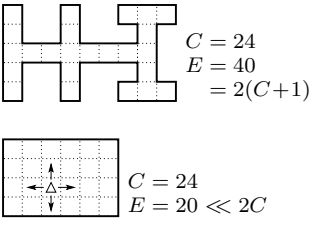


Fig. 1. The perimeter, E, to distinguish between 'thin' and 'thick' environments.

Our paper is organized as follows: in Sect. 2 we give more detailled description of our robot and the environment. We give a lower bound for our problem in Sect. 3. In Sect. 4 we present an exploration strategy, SmartDFS. The analysis shows in Sect. 5 that this strategy uses no more than $C+\frac{1}{2}E-3$ steps and is in fact competitive with a factor of $\frac{4}{3}$.

SmartDFS was implemented in a Java-Applet available in the internet, see [8].

[1] We assume that the cells have unit size, so the length of the path is equal to the number of steps from cell to cell.

2 Definitions

We consider a simple model for the environment of the robot: the robot moves in a surrounding with a grid structure. More precisely, a *cell* is a basic block in our environment, defined by a pair $(x, y) \in \mathbb{N}^2$. A cell is either *free* and can be visited by the robot, or *blocked*, i. e., unaccessible for the robot. We call two cells *adjacent*, if they share a common edge, and *touching*, if they share a common edge or corner. A *grid polygon*, P, is a connected set of free cells. A polygon without blocked cells inside its boundary is called *simple*. From its current position, the robot can find out which of the adjacent cells are free and which are blocked, and it can move in one *step* to an adjacent free cell, see Fig. 1. The robot has enough memory to store a map of known cells.

3 A Lower Bound

Theorem 1. *Every strategy for the exploration of a simple grid polygon with C cells needs at least $\frac{7}{6}C$ steps.*

Proof. We assume that the robots starts in a corner of the polygon, see Fig. 2(i). W. l. o. g. we assume that the strategy decides to walk one step to the east. For the second step, the strategy has two possibilities: either it leaves the wall with a step to the south, see Fig. 2(ii), or it continues to follow the wall with a further step to the east, see Fig. 2(iii). In the first case, we close the polygon as shown in Fig. 2(iv). The robot needs at least 8 steps to explore this polygon, but the optimal strategy needs only 6 steps yielding a factor of $\frac{8}{6}$. In the second case we proceed as follows. If the robot leaves the boundary, we close the polygon as shown in Fig. 2(v) and (vi). The robot needs 12 step, but 10 steps are sufficient. In the most interesting case, the robot still follows the wall, see Fig. 2(vii). In this case, the robot needs at least 28 steps to explore this polygon, whereas an optimal strategy needs only 24 steps. Thus, we achieve a factor of $\frac{7}{6}$.

We can easily extend this pattern to polygons of arbitrary size by repeating the construction using the 'entry' and 'exit' cells denoted by the arrows in Fig. 2(iv)–(vii). This construction cannot lead to overlapping polygons or polygons with holes, since the polygon always extends to the same direction. □

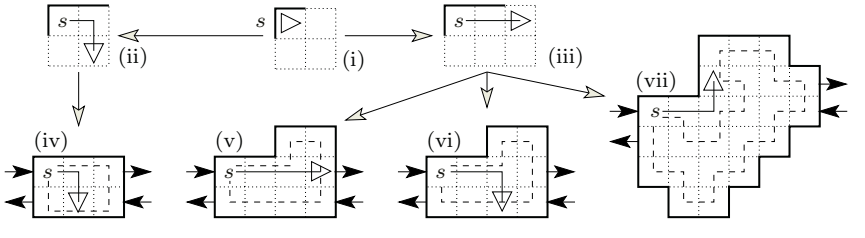


Fig. 2. A lower bound for the exploration of simple polygons. The dashed lines show the optimal solution, △ denotes the robot's position.

4 An Exploration Strategy

As a first approach, we can apply a simple depth-first search algorithm (DFS): The polygon is explored following the left-hand rule, i. e., for every entered cell the robot tries to continue its path to an adjacent and unexplored cell, preferring a step to the left over a straight step over a step to the right. This results in a complete exploration, but takes $2C - 2$ steps. Since the shortest tour needs at least C steps, DFS turns out to be 2-competitive. However, there is no reason to visit *each* cell twice just because this is required in some special situations like dead ends of width 1. In the following, we introduce two improvements to DFS.

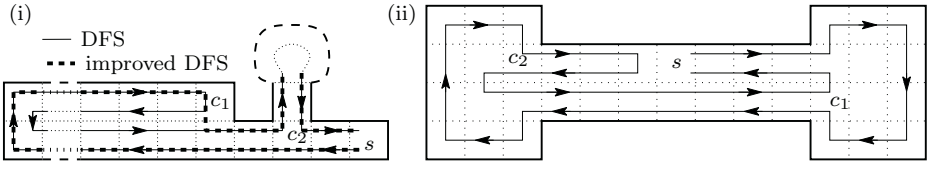


Fig. 3. Improvement to DFS: (i) optimize return path, (ii) detect polygon splits.

The first improvement is to return directly to those cells that have unexplored neighbors. See e. g. Fig. 3(i): DFS walks from c_1 to c_2 through the completely explored corridor. A more efficient strategy walks on a shortest path – on cells that are already known – from c_1 to c_2.

Now, observe the polygon shown in Fig. 3(ii). With DFS, the robot walks four times through the narrow corridor. A more clever solution explores the right part immediately after the first visit of c_1, and continues with the left part, resulting in only two visits. The cell c_1 has the property that the graph of unvisited cells splits into two components after c_1 is explored. We call cells like this *split cells*. The second improvement is to recognize and handle split cells, see Sect. 5. The following description of our strategy, SmartDFS, resumes both improvements to DFS, see Fig. 5 for an example.

SmartDFS(*P*, *start***):**
Choose direction *dir*, such that reverse(*dir*) is a blocked cell;
ExploreCell(*dir*);
Walk on the shortest path to *start*;

ExploreStep(*base*, *dir***):**
if unexplored(*base*, *dir*) **then**
Walk on shortest path to *base*;
move(*dir*);
ExploreCell(*dir*);
end if

ExploreCell(*dir***):**
base := current position;
if not isSplitCell(*base*) **then**
ExploreStep(*base*, ccw(*dir*));
ExploreStep(*base*, *dir*);
ExploreStep(*base*, cw(*dir*));
else
Choose different order, see Sect. 5.
end if

5 The Analysis of SmartDFS

SmartDFS explores the polygon in layers, beginning with the cells along the boundary of P and proceeding towards the interior of P.

Definition 2. *Let P be a grid polygon. The boundary cells of P uniquely define the* first layer *of P. The polygon P without its first layer is called the* 1-offset *of P. The ℓ-th layer and the ℓ-offset of P are defined successively, see Fig. 4(i).*

Lemma 3. *The ℓ-offset of a simple grid polygon, P, has at least 8ℓ edges less than P.*

Proof. First, we cut off blind alleys narrower than 2ℓ, since those parts of P do not affect the ℓ-offset. We walk clockwise around the boundary cells of the remaining polygon, see Fig. 4(i). For every left turn the offset gains at most 2ℓ edges and for every right turn the offset looses at least 2ℓ edges. Since, there are four more right turns than left turns, we loose at least 8ℓ edges. □

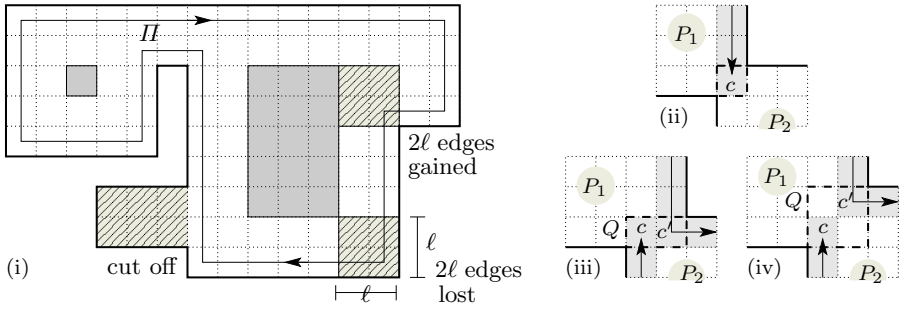


Fig. 4. (i) The 2-offset (shaded) of a grid polygon; three examples for split cells, (ii) type (II), (iii) and (iv) type (I).

Definition 2 allows us to specify the handling of a split cell in SmartDFS. Let us consider the situation shown in Fig. 5(i): SmartDFS has just met the first split cell, c, in the fourth layer of P. P divides into three parts: $P = K_1 \dot\cup K_2 \dot\cup \{\text{ visited cells of } P\}$, where K_1 and K_2 denote the connected components of the unvisited cells. In this case it is reasonable to explore the component K_2 first since the start cell s is closer to K_1.

We use the layer numbers to decide which component we have to visit at last. Whenever a split cell occurs in layer ℓ, every component is one of the following types, see Fig. 4(ii)–(iv): (I) K_i is *completely* surrounded by layer ℓ [2], (II) K_i is *not* surrounded by layer ℓ, or (III) K_i is *partially* surrounded by layer ℓ.

In any case, it is the best choice to explore the component of type (III) at last. Note that it may occur that three components arise at a split cell, but we can handle this case as two successive splits occuring at the same split cell.

[2] More precisely, the part of layer ℓ that surrounds K_i is completely visited. For convenience, we will use slightly sloppy, but shorter form.

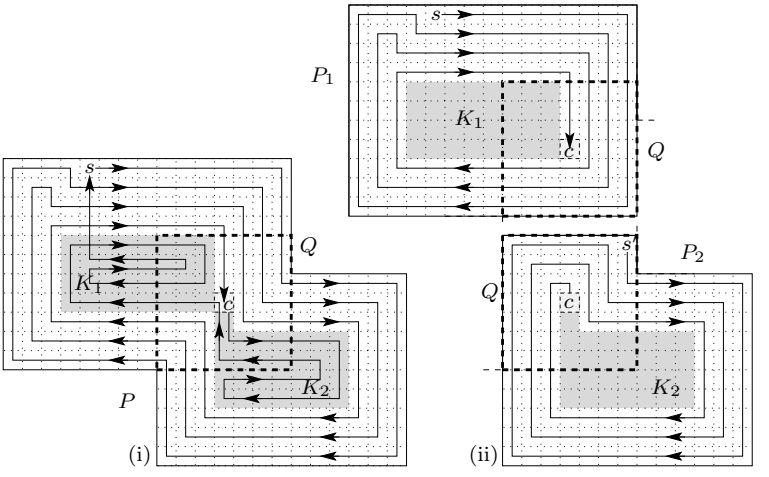


Fig. 5. A decomposition of P at the split cell c and its handling in smartDFS.

For the analysis we consider two polygons, P_1 and P_2, as follows. Let Q be the square of width $2q+1$ around c with $q := \begin{cases} \ell, & \text{if } K_2 \text{ is of type (I)} \\ \ell - 1, & \text{if } K_2 \text{ is of type (II)} \end{cases}$, where K_2 denotes the component that is explored first, and ℓ denotes the layer in which the split cell was found. We choose $P_2 \subset P \cup Q$, such that $K_2 \cup \{c\}$ is the q-offset of P_2, and $P_1 := ((P \backslash P_2) \cup Q) \cap P$, see Fig. 5. The intersection with P is necessary, since Q may exceed the boundary of P.

The choice of P_1, P_2 and Q ensures that the robot's path in $P_1 \backslash Q$ and in $P_2 \backslash Q$ do not change compared to the path in P. The parts of the robot's path that lead from P_1 to P_2 and from P_2 to P_1 are fully contained in the square Q. Just the parts inside Q are bended to connect the appropriate paths inside P_1 and P_2, see Fig. 5.

We want to visit every cell in the polygon and to return to s. Every strategy needs at least $C(P)$ steps to fulfill this task. Thus, we can split the overall length of the exploration path Π into two parts, $C(P)$ and excess(P), with $|\Pi| = C(P) + \text{excess}(P)$. Since SmartDFS recursively explores $K_2 \cup \{c\}$, we want to apply the upper bound inductively to the component $K_2 \cup \{c\}$. The following lemma gives us the relation between the path lengths in P and the path lengths in the two components.

Lemma 4. *Let P be a simple grid polygon. Let the robot visit the first split cell, c, which splits the unvisited cells of P into two components K_1 and K_2, where K_2 is of type (I) or (II). With the preceding notions we have* $\text{excess}(P) \leq \text{excess}(P_1) + \text{excess}(K_2 \cup \{c\}) + 1$.

Proof. Since c is the first split cell, there is no excess in $P_2 \backslash (K_2 \cup \{c\})$ and it suffices to consider excess$(K_2 \cup \{c\})$ for this part. After $K_2 \cup \{c\}$ is finished, the robot starts at c and explores K_1. For this part we take excess(P_1) into account. Finally, we add one single step, because the split cell c is visitited twice:

once, when SmartDFS detects the split and once more after the exploration of excess($K_2 \cup \{c\}$) is finished. Altogether, the given bound is achieved. □

The following lemma can easily be shown and allows us to charge the number of edges in P_1 and P_2 against the number of edges in P and Q.

Lemma 5. *Let P be a simple grid polygon, and let P_1, P_2 and Q be defined as above. The number of edges satisfy $E(P_1) + E(P_2) = E(P) + E(Q)$.*

Lemma 6. *Let Π be the shortest path between two cells in a grid polygon P. The length of Π is bounded by $|\Pi| \leq \frac{1}{2}E(P) - 2$.*

Proof. The maximal distance is achieved between two cells in the first layer, and the shortest path between them is never longer than $\frac{1}{2}\cdot\#$(cells in the first layer). Analogously to Lemma 3, this layer has at most $E(P) - 4$ cells. □

Now, we can give an upper bound for the number of steps used to explore a simple polygon.

Theorem 7. *Let P be a simple grid polygon with C cells and E edges. P can be explored with $S \leq C + \frac{1}{2}E - 3$ steps. This bound is tight.*

Proof. C is the number of cells and thus a lower bound on the number of steps that are needed to explore the polygon P. We will show by induction on the number of components that excess(P) $\leq \frac{1}{2}E(P) - 3$ holds.

For the induction base we consider a polygon without any split cell, i. e., SmartDFS visits all cells and returns on the shortest path to the start cell. Since there is no polygon split, all cells of P can be visited by a path of length $C - 1$. By Lemma 6 the shortest path back to the start cell is not longer than $\frac{1}{2}E - 2$ and excess(P) $\leq \frac{1}{2}E(P) - 3$ holds.

Now, we assume that there is more than one component during the application of SmartDFS. Let c be the first split cell detected in P. When SmartDFS reaches c, two new components, K_1 and K_2, occur. We consider the two polygons P_1 and P_2 defined as above using the square Q around c.

W. l. o. g. we assume that K_2 is recursively explored first. After K_2 is completely explored, SmartDFS proceeds with the remaining polygon. As shown in Lemma 4 we have excess(P) $\leq$ excess(P_1) + excess($K_2 \cup \{c\}$) + 1 . Now, we apply the induction hypothesis to P_1 and $K_2 \cup \{c\}$ and get

$$\text{excess}(P) \leq \tfrac{1}{2}E(P_1) - 3 + \tfrac{1}{2}E(K_2 \cup \{c\}) - 3 + 1\,.$$

By applying Lemma 3 to the q-offset $K_2 \cup \{c\}$ of P_2 we achieve

$$\text{excess}(P) \leq \tfrac{1}{2}E(P_1) - 3 + \tfrac{1}{2}(E(P_2) - 8q) - 3 + 1 = \tfrac{1}{2}(E(P_1) + E(P_2)) - 4q - 5\,.$$

From Lemma 5 we conclude $E(P_1) + E(P_2) \leq E(P) + 4(2q + 1)$. Thus, we get excess($P$) $\leq \frac{1}{2}E(P) - 3$. This bound is achieved exactly in polygons that do not contain any 2×2-square of free cells. □

So far we have shown an upper bound for the number of steps needed to explore a polygon that depends on the number of cells and edges in the polygon. Now we want to analyze SmartDFS in the competitive framework.

Corridors of width 1 or 2 play a crucial role in the following, so we refer to them as *narrow passages*[3]. It is easy to see that narrow passages are explored optimally. In passages of width 1 both SmartDFS and the optimal strategy visit every cell twice, and in the other case both strategies visit every cell exactly once. We need two lemmata to show a competitive factor for SmartDFS. The first one gives us a relation between the number of cells and the number of edges for a special class of polygons.

Lemma 8. *For a simple grid polygon, P, without any narrow passage or split cells in the first layer, $E(P) \leq \frac{2}{3}C(P) + 6$ holds.*

Proof. Consider such a polygon, P, see Fig. 6(i). We successively remove an outer row or column of at least three boundary cells, maintaining our assumptions on P. These assumptions ensure that we can always find such a row or column. Thus, we remove at least three cells and at most two edges. This decomposition ends with a 3×3 block of cells that fulfills $E = \frac{2}{3}C(P) + 6$. Now, we reverse our decomposition, i. e., we successively add all rows and columns until we end up with P. In every step, we add at least three cells and at most two edges. Thus, $E \leq \frac{2}{3}C(P) + 6$ is fulfilled in every step. □

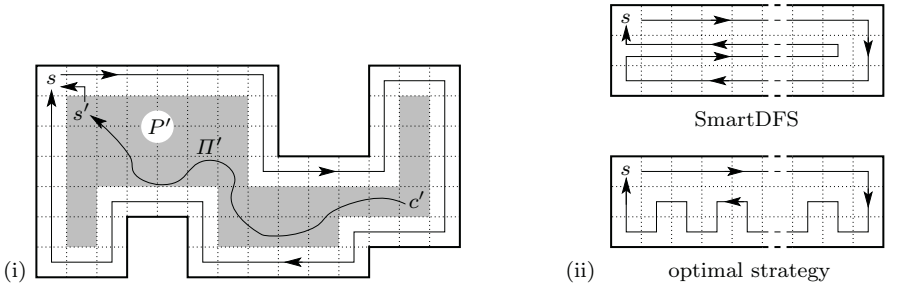


Fig. 6. (i) For polygons without narrow passages or split cells in the first layer, $E(P) \leq \frac{2}{3}C(P) + 6$ holds, and the last explored cell, c', lies in the 1-offset, P' (shaded), (ii) In a corridor of width 3 and even length, $S(P) = \frac{4}{3}S_{\mathrm{Opt}}(P) - 2$ holds.

For the same class of polygons, we can show that SmartDFS behaves slightly better than the bound in Theorem 7.

Lemma 9. *A polygon of the same type as in Lemma 8 can be explored using no more than $S(P) \leq C(P) + \frac{1}{2}E(P) - 5$ steps.*

Proof. We have shown $S(P) \leq C(P) + \frac{1}{2}E(P) - 3$ in Theorem 7. In the proof, we used Lemma 6 to bound the return path, but this lemma bounds the path between two cells in the first layer. By our assumptions on P, we can completely explore the first layer of P before visiting another layer, and the return path, Π, starts in a cell, c', in the 1-offset, P', see Fig. 6(i). Let s' denote the first

[3] More precisely, a cell, c, belongs to a narrow passage, if c can be removed without changing the layer number of any other cell.

visited cell in P'. Remark that s and s' are at least touching each other. Now, Π is bounded by a shortest path, Π', from c' to s' in P' and a shortest path from s' to s, i. e., $|\Pi| \leq |\Pi|' + 2$. Π', in turn, is bounded using Lemma 6 by $|\Pi|' \leq E(P') - 2$. With Lemma 3, $E(P') \leq E(P) - 4$ holds, and altogether we get $|\Pi| \leq E(P) - 4$, which is two steps shorter than stated in Lemma 6. □

Theorem 10. *The strategy* SmartDFS *is* $\frac{4}{3}$*-competitive.*

Proof. Let P be a simple grid polygon. First, we remove all narrow passages from P and get a sequence of (sub-)polygons P_i, $i = 1, \ldots, k$, without narrow passages. For every P_i, $i = 1, \ldots, k-1$, the optimal strategy in P explores the part of P that corresponds to P_i up to the narrow passage that connects P_i with P_{i+1}, enters P_{i+1}, and fully explores every P_j with $j \geq i$. Then it returns to P_i and continues with the exploration of P_i. Further, we already know that narrow passages are explored optimally. This allows us to consider every P_i separately without changing the competitive factor of P.

Now, we observe a (sub-)polygon P_i. We show by induction on the number of split cells in the first layer that $S(P_i) \leq \frac{4}{3}C(P_i) - 2$ holds. Note that this is exactly achieved in polygons of size $3 \times m$ with m even, see Fig. 6(ii).

If P_i has no split cell in the first layer, we can apply Lemma 9 and Lemma 8:
$S(P_i) \leq C(P_i) + \frac{1}{2}E(P_i) - 5 \leq C(P_i) + \frac{1}{2}\left(\frac{2}{3}C(P_i) + 6\right) - 5 = \frac{4}{3}C(P_i) - 2\,.$

Two cases occur if we meet a split cell, c, in the first layer, see Fig. 4(ii)–(iv). In the first case, the new component was never visited before (type (II)). Here, we define $Q := \{c\}$. The second case occurs, because the robot meets a cell, c', that is in the first layer and touches the current cell, c, see for example Fig. 4(iii) and (iv). Let Q be the smallest rectangle that contains both c and c'.

Similar to the proof of Theorem 7, we split the polygon P_i into two parts, both including Q. Let P'' denote the part that includes the component of type (II) or (III), P' the other part. For $|Q| = 1$, see Fig. 4(ii), we conclude $S(P_i) = S(P') + S(P'')$ and $C(P_i) = C(P') + C(P'') - 1$. Applying the induction hypothesis to P' and P'' yields $S(P_i) = S(P') + S(P'') \leq \frac{4}{3}C(P_i) + \frac{4}{3} - 4 \;<\; \frac{4}{3}C(P_i) - 2\,.$

For $|Q| \in \{2, 4\}$ we gain some steps by merging the polygons. If we consider P' and P'' separately, we count the steps from c' to c – or vice versa – in both polygons, but in P_i the path from c' to c is replaced by the exploration path in P''. Thus, we have $S(P_i) = S(P') + S(P'') - |Q|$ and $C(P_i) = C(P') + C(P'') + |Q|$. This yields $S(P_i) = S(P') + S(P'') - |Q| = \frac{4}{3}C(P_i) + \frac{1}{3}(|Q| - 6) - 2 < \frac{4}{3}C(P_i) - 2\,.$

An optimal strategy needs $\geq C$ steps, which, altogether, yields a competitive factor of $\frac{4}{3}$. □

6 Summary

It turned out that the exploration of simple polygons is easier than the exploration of polygons with holes in terms of competitivity. In contrary to the lower bound of 2 for polygons with holes, we have shown a lower bound of $\frac{7}{6}$ and an upper bound of $\frac{4}{3}$ for simple polygons, leaving a gap of only $\frac{1}{6}$. Additionally, we can also bound the length of an exploration path by $C + \frac{1}{2}E - 3$ which is tight.

References

1. E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell. Approximation algorithms for lawn mowing and milling. Technical report, Mathematisches Institut, Universität zu Köln, 1997.
2. S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 2–11, 1996.
3. X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment I: The rectilinear case. *J. ACM*, 45(2):215–245, 1998.
4. H. Everett. Hamiltonian paths in non-rectangular grid graphs. Report 86-1, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, 1986.
5. A. Fiat and G. Woeginger, editors. *On-line Algorithms: The State of the Art*, volume 1442 of *Lecture Notes Comput. Sci.* Springer-Verlag, 1998.
6. Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Comput. Geom. Theory Appl.*, 24:197–224, 2003.
7. M. Grigni, E. Koutsoupias, and C. H. Papadimitriou. An approximation scheme for planar graph TSP. In *Proc. 36th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 640–645, 1995.
8. U. Handel, C. Icking, T. Kamphans, E. Langetepe, and W. Meiswinkel. Gridrobot – an environment for simulating exploration strategies in unknown cellular areas. Java Applet, 2000. http://www.geometrylab.de/Gridrobot/.
9. F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM J. Comput.*, 31:577–600, 2001.
10. C. Icking, T. Kamphans, R. Klein, and E. Langetepe. Exploring an unknown cellular environment. In *Abstracts 16th European Workshop Comput. Geom.*, pages 140–143. Ben-Gurion University of the Negev, 2000.
11. C. Icking, T. Kamphans, R. Klein, and E. Langetepe. On the competitive complexity of navigation tasks. In H. Bunke, H. I. Christensen, G. D. Hager, and R. Klein, editors, *Sensor Based Intelligent Robots*, volume 2238 of *Lecture Notes Comput. Sci.*, pages 245–258, Berlin, 2002. Springer.
12. C. Icking, R. Klein, and E. Langetepe. Searching for the kernel of a polygon: A competitive strategy using self-approaching curves. Technical Report 211, Department of Computer Science, FernUniversität Hagen, Germany, 1997.
13. C. Icking, R. Klein, E. Langetepe, S. Schuierer, and I. Semrau. An optimal competitive strategy for walking in streets. *SIAM J. Comput.*, 33:462–486, 2004.
14. A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Comput.*, 11:676–686, 1982.
15. T. Kamphans. *Models and Algorithms for Online Exploration and Search*. PhD thesis, University of Bonn, to appear.
16. J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28:1298–1309, 1999.
17. J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.
18. S. Ntafos. Watchman routes under limited visibility. *Comput. Geom. Theory Appl.*, 1(3):149–170, 1992.
19. C. Umans and W. Lenhart. Hamiltonian cycles in solid grid graphs. In *Proc. 38th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 496–507, 1997.

Approximation Algorithms for Cutting Out Polygons with Lines and Rays

Xuehou Tan

Tokai University, 317 Nishino, Numazu 410-0395, Japan
tan@wing.ncc.u-tokai.ac.jp

Abstract. This paper studies the problem of cutting out a given polygon, drawn on a convex piece of paper, in the cheapest possible way. For the problems of cutting out convex polygons with line cuts and ray cuts, we present a 7.9-approximation algorithm and a 6-approximation algorithm, respectively. For the problem of cutting out ray-cuttable polygons, an $O(\log n)$-approximation algorithm is given.

1 Introduction

Overmars and Welzl introduced the following problem about two decades ago: Given a polygonal piece of paper Q with a polygon P drawn on it, cut P out of Q in a cheapest possible way [5].

A *cut* is a line that does not intersect the interior of P and divides Q into a number of pieces, lying on both sides of the line. The cost of a cut is the length of the intersection of the cut with Q. A cut is an *edge cut* if it cuts along an edge of P. After a cut is made, the polygon Q is updated to the piece containing P. A *cutting sequence* is a sequence of cuts such that after the last cut in the sequence we have $P = Q$. One natural question is to find an optimal cutting sequence whose total cost is minimum.

It is clear that the problem is solvable only when P is convex. If Q is also convex, then there exists an optimal cutting sequence with $O(n)$ line cuts, in which each cut touches P [5]. However, the problem has optimal solutions that may lie in the algebraic extension of the field that the input data belongs to [1]. Thus, an approximation scheme is the best one can achieve.

The first polynomial-time approximation algorithm is due to Dumitrescu [4]. The solution consists of a *separation phase* and a *carving phase*. In the separation phase, three line cuts are used to cut out a triangle that contains P and has roughly the same size of P. In the carving phase, the polygon P is cut out by a sequence of edge cuts of cost no more than $O(\log n)$ times the perimeter of P, where n is the number of vertices of P. This gives an $O(\log n)$-approximation algorithm with $O(mn + n \log n)$ running time, where m is the number of vertices of Q [4]. Very recently, Daescu and Luo showed that if P is enclosed in a minimum axis-aligned rectangle Q, an optimal edge cutting sequence gives a $(2.5 + \sqrt{2})$-approximation [2]. Combining with the separation phase given by Dumitrescu, it leads to an $O(1)$-approximation algorithm with $O(n^3 + (n + m) \log(n + m))$ running time [2]. When only edge cuts are allowed, an optimal cutting sequence can be found in $O(n^3 + m)$ time using a dynamic programming algorithm [5].

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 534–543, 2005.

Daescu and Luo also considered the problem of cutting out polygons with ray cuts. A *ray cut* originates from infinity and ends at some point within Q. Demaine et al. have presented a linear time algorithm to determine if a polygon is *ray-cuttable* [3]. For the problem of cutting out convex polygons, Daescu and Luo have given an 18-approximaton algorithm to cut out a triangle that contains P. For the problem of cutting out ray-cuttable polygons, they have presented an $O(\log^2 n)$-approximation algorithm with $O(m + n \log n)$ running time [2].

Our Results. In this paper, we first present an $O(n + m)$ time algorithm to cut out a minimum axis-aligned rectangle that contains the convex polygon P, at the cost no more than $(3.5 + \sqrt{2})$ times that of the optimal cutting sequence. Next, we show that if P is enclosed in a minimum rectangle, an optimal edge cutting sequence is a $(1.5 + \sqrt{2})$-approximation of an optimal cutting sequence. It thus results in a $(5 + 2\sqrt{2})$-approximation, or shortly, a 7.9-approximation algorithm with $O(n^3 + m)$ running time. Alternatively, we give an $O(\log n)$-approximation algorithm that requires only $O(n+m)$ time, an impovement over the solution in [2] by a factor $\log(n+m)$. For the problem of cutting out a convex polygon P with ray cuts, we also give a $(5/\sqrt{2})$-approximation algorithm to cut out a minimum rectangle that contains P. In the carving phase, we present a dynamic programming algorithm to compute an optimal ray cutting sequence such that all cuts in the sequence are along the edges of P, and then show that the found sequence is a $(1 + \sqrt{2})$-approximation. Thus, we obtain a $(1 + 7/\sqrt{2})$-approximation, or shortly, a 6-approximation algorithm with $O(n^3 + m)$ running time. For the problem of cuttting out ray-cuttable polygons, we present an $O(\log n)$-approximation algorithm with $O(n^3 + m)$ running time.

2 Cutting Out Convex Polygons with Line Cuts

Let P, Q be two convex polygons such that $P \subset Q$, and ∂P (resp. ∂Q) the boundary of P (resp. Q). Let S^* denote an optimal cutting sequence, and S_e^* an optimal edge cutting sequence. The cost of a cutting sequence S is denoted by $|S|$. Let ab denote a line segment with endpoints a and b. We denote by $|ab|$ the length of ab. Also, we use the notation $|P|$ for the perimeter of P, and use D for the diameter of P. Clearly, $2D \leq |P| \leq |S^*|$ holds [2–4].

2.1 Separation Phase

Dumitrescu was the first to give an $O(nm)$ time algorithm to find a triangle that contains P and has roughly the same size of P [4]. For a vertex v of P, define the *chord through* v as the intersecting segment of Q with a line through v. The angles between all pairs of shortest chords are examined by his algorithm. In this section, we show that two shortest chords suffice to bound the size of the minimum rectangle containing P. It results in not only a good approximation factor, but also a simple algorithm for cutting out the rectangle containing P.

Let C_v denote a shortest chord through a vertex v of P. Let $V(P)$ be the set of vertices of P, and let $|C| = max_{v \in V(P)} |C_v|$. Then, $|C| \leq |S^*|$ holds [4].

A chord through $v \in V(P)$ is called an *ST-chord* if it is a shortest chord through v and tangent to P. We denote it by ST_v. Let $|ST| = min_{v \in V(P)}|ST_v|$.

Lemma 1 *Let T denote the vertex or the edge of P that is touched by the chord ST. Then, $|ST| + |P| - |T| \leq |S^*|$ holds.*

Proof. Since there is an optimal cutting sequence S^* such that all cuts in S^* touch P [5], the length of the first cut in S^* is at least $|ST|$. All edge cuts have to appear in S^*; otherwise, P cannot be cut out. Hence, the lemma follows. □

W.l.o.g., assume that the chord ST is on the x-axis. Let Q' denote the minimum rectangle containing P. Then, $|Q'| \leq \sqrt{2}|P| \leq \sqrt{2}|S^*|$ holds. Let D_x (resp. D_y) denote the x-distance (resp. y-distance) between two vertical (resp. horizontal) edges of Q'. Then, $D_x \leq D$, $D_y \leq D$, and $D_x + D_y \leq \sqrt{2}|S^*|/2$ hold.

Theorem 1 *There is an $O(n+m)$ time algorithm that cuts out a minimum rectangle containing P, through a cutting sequence of cost less than $(3.5 + \sqrt{2})|S^*|$.*

Proof. Suppose that a chord ST touches a vertex v of P. Let l_v denote the supporting line of the chord ST. Our first cut is made along l_v. See Fig. 1, where the cuts are shown with bold and dotted lines. Let u be the vertex of P whose distance to l_v is maximum, and u' the point of l_v such that the angle $\angle uu'v$ is $\pi/2$. Denote by C_u a shortest chord through u, and d, e two endpoints of C_u on ∂Q. Let l_u and l_e denote the lines parallel to l_v and through u and e, respectively. W.l.o.g., assume that the point e lies in between l_u and l_v. (The point e may be on l_v.) Let f denote the intersection point of l_v with the supporting line of de. We distinguish below the following different cases.

Case 1 The chord C_u is tangent to P. In this case, we make the second cut along C_u, whose length is at most $|S^*|$. W.l.o.g., assume that the slopes of the first two cuts are different. The lines through two cuts ST and C_u partition the plane into four wedges. Define the angle of the wedge containing P as the *enclosing angle* of ST and C_u.

Case 1.1 The enclosing angle of ST and C_u is at most $\pi/2$. Consider a line segment ap tangent to P, with p on l_v and a on the semi-line originating from u' and going through u. (The point p may not lie in Q.) Let b be the intersection point of ap with l_u. Assume also that b and e lie in different sides of au' (Fig. 1a). Our third cut is made along ap, with the point a giving the minimum value $|ap|_{min}$ of $|ap|$. (The point a giving $|ap|_{min}$ may not lie in Q.)

Let us now give an upper bound on $|ap|_{min}$. Set $Y = |au|$. Since $|au'| = |au| + |uu'| \leq Y + D_y$ and $|ab| < |au| + |bu| = Y + |bu|$, we have $|ap| = |au'|(|ab|/|au|) = (D_y/Y + 1)(Y + |bu|)$. Since $|bu| \leq D_x$ holds, we consider the function $F(Y) = (D_y/Y + 1)(Y + D_x)$. A simple analysis shows that the minimum value $F(Y)_{min}$ of $F(Y)$ is achieved when $Y = \sqrt{D_x D_y}$. Hence,

$$F(Y)_{min} < (D_y/\sqrt{D_x D_y} + 1)(\sqrt{D_x D_y} + D_x) \leq (1 + \sqrt{2}/2)|S^*|.$$

It is clear that $|ap|_{min} \leq F(Y)_{min}$. Thus, the length of the cut along ap is less than $(1+\sqrt{2}/2)|S^*|$. Finally, we make a cut along l_u, and two cuts parallel to au'

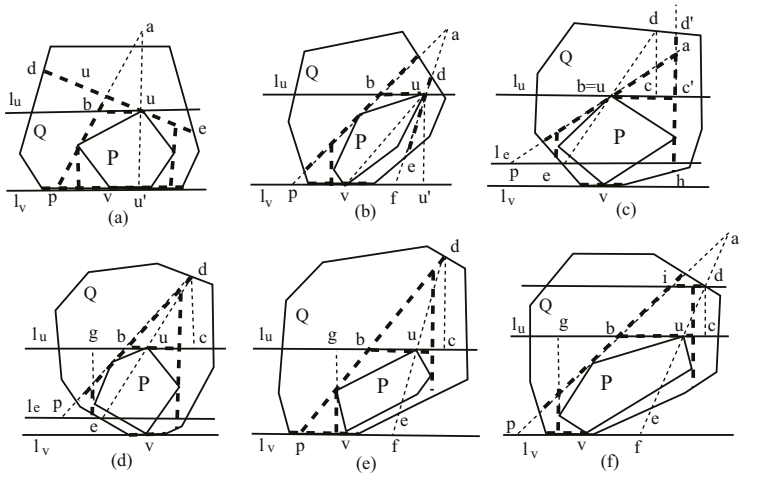


Fig. 1. Illustration of cutting out a minimum rectangle containing P.

and tangent to P. This results in the minimum rectangle Q' containing P (only part of Q' may result). See Fig. 1a. The sum of the cost taken for the cut along the chord ST and the costs for the last three cuts is less than $|Q'| + |ST| - |T|$. Since $|ST| \geq |T|$, we have $|Q'| + |ST| - |T| \leq \sqrt{2}(|P| + |ST| - |T|) \leq \sqrt{2}|S^*|$. Hence, the total cost of these six cuts is less than $(2 + \frac{3}{\sqrt{2}})|S^*|$.

Case 1.2 The enclosing angle of ST and C_u is larger than $\pi/2$ (Fig. 1b). Consider a line segment ap tangent to P, with the point p on l_v and the point a on the semi-line originating from f and going through d, such that $|ap| > |af|$. Let b be the intersection point of ap with l_u. See Fig. 1b.

Set $Y = |au|$. Since the point f is contained in the right-angled triangle $\triangle_{uu'v}$ in this case, $|uf| < |uv| \leq D$ holds. Thus, $|af| = |au| + |uf| \leq Y + D$. Since $|bu| \leq D_x$ holds, we have $|ab| < |au| + |bu| \leq Y + D_x$. Hence, $|ap| = |af|(|ab|/|au|) < (D + Y)(Y + D_x)/Y \leq (D/Y + 1)(Y + D_x)$.

Also, the following upper bound on the minimum value $|ap|_{min}$ of $|ap|$ holds.

$$|ap|_{min} < D + 2\sqrt{DD_x} + D_x \leq |S^*|/2 + |S^*| + |S^*|/2 = 2|S^*|.$$

To cut out the rectangle Q', we compute the position of a giving $|ap|_{min}$, and make the third cut along ap. Finally, make a cut along l_u and a cut tangent to P and perpendicular to l_v. The sum of the costs for the cut along ST and for the last two cuts is at most $\sqrt{2}|S^*|$. The total cost is less than $(3 + \sqrt{2})|S^*|$.

Case 2 The chord C_u intersects the interior of P. Let c be the point of l_u such that the angle $\angle dcu$ is $\pi/2$.

Case 2.1 The supporting line of cd intersects the interior of P (Fig. 1c). Because of the convexity of Q, we can move a line segment $c'd'$, with c' on l_u and d' on ∂Q, to a place such that the supporting line of $c'd'$ is parallel to cd and tangent to P, and $|c'd'| \leq |cd|$. See Fig. 1c. In this case, the second cut is made along the supporting line of $c'd'$. Denote by h the other intersection of this

cut with ∂Q. For efficiency, the cost of our second cut is considered as two parts $|c'h|$ and $|c'd'|$. Note that $|c'd'| \leq |cd| < |ud| < C_u \leq |S^*|$.

Consider a line segment ap tangent to P, with p on the line l_e and a on the semi-line originating from h and going through d'. Let b be the intersection point of the line l_u with ap. Then, $|bc'| \leq D_x$ holds. See Fig. 1c. Again, we compute the position of a that gives the minimum value of $|ap|$, and make the third cut along ap. The rectangle Q' can finally be cut out by making two cuts along l_u and along the other tangent to P which is parallel to $d'h$. The sum of $|c'h|$ and the costs taken for the first cut ST and the last two cuts is at most $|Q'| + ST - T \leq \sqrt{2}|S^*|$. As in Case 1.1, the total cost is less than $(2 + \frac{3}{\sqrt{2}})|S^*|$.

Case 2.2 The supporting line of cd does not intersect the interior of P. Let g denote the vertex of the rectangle Q' such that it is on l_u and $|dg| > |du|$.

Case 2.2.1 $|gu| \leq |du|$. Suppose first that d and e are to different sides of the vertex v (as viewed from v). See Fig. 1d. Let dp denote the segment tangent to P, with p on l_e, such that $|dp| > |de|$. The second cut is made along the supporting line of dp. Let b be the intersection point of the cut with l_u. Since $|bu| \leq |gu|$, we have $|dp| = |de|(|db|/|du|) < |de|(|bu| + |du|)/|du| \leq |de|(|gu|/|du| + 1) \leq 2|de|$.

To cut out the rectangel Q', we make two cuts tangent to P and perpendicular to l_v. Also, the longer cut is considered as two parts: one is between l_v and l_u, and the other is between l_u and the line segment bd, whose length is less than $|dc| < |du| < C_u < |S^*|$. See Fig. 1d. Finally, make a cut along l_u. As analyzed above (like Case 1.2), the total cost is then less than $(3 + \sqrt{2})|S^*|$.

Assume now that d and e are to the same side of v (Fig. 1e). Let dp denote the segment tangent to P, with p on l_v, such that $|dp| > |df|$. Note that $|df| = |de| + |ef| \leq |S^*| + |ef|$. Our cutting sequence is exactly the same as above, except for that the cost for the cut along dp is increased by $2|ef|$. Since the angle $\angle efv$ is larger than $\pi/2$ in this case, $|ef| < |ev|$. Since ev is a part of the final polygon Q and does not overlap with any cut, a mount of $|ef|$ can be saved. Since $|ef| < |uf| < |uv| < D$, the total cost is less than $(3.5 + \sqrt{2})|S^*|$.

Case 2.2.2 $|gu| > |du|$ *(Fig. 1f).* Assume first that d and e are to the same side of the vertex v. Let ap be a line segment tangent to P, with p on l_v and a on the semi-line originating from f and going through d, such that $|ap| > |af|$. Let b and i be the intersection points of ap with l_u and with the line through d and parallel to l_v, respectively. See Fig. 1f.

Set $Y = |ad|$ and $H = |df|$. Since $|uf| < |uv|$ holds in this case, $H = |du| + |uf| < |gu| + |uv| \leq D_x + D \leq |S^*|$. Since $|af| = |ad| + |df| = Y + H$ and $|ai| < |ad| + |di| = Y + |di|$, we have $|ap| = |af|(|ai|/|ad|) < (H/Y + 1)(Y + |di|)$. Since $|di| < |bu| \leq D_x$ holds, we have $|ap|_{min} < H + 2\sqrt{HD_x} + D_x \leq (1.5 + \sqrt{2})|S^*|$.

To cut out the rectangle Q', we compute the position of a giving $|ap|_{min}$, and make the third cut along ap. Next, make a cut along di, whose length is less than $|bu|$. Two cuts, tangent to P and perpendicular to l_v, are then made. Note that the longer cut can be considered as two parts: one is between l_v and l_u and the other is between l_u and the supporting line of di (whose length is less than $|gu|$). Finally, a cut along l_u is made. The sum of $|gu|$ and the cost of the cut along l_u is clearly less than $2D_x \leq |S^*|$. Consider now the costs represented by $|bu|$ and taken for the first cut along ST and two cuts that are tangent to

P and perpendicular to l_v, excluding the part of the longer cut between l_u and the supporting line of di. By noticing the relation between the segment bu and two considered vertical cuts (Fig. 1f), we have that the sum of these costs is less than $|P| + |ST| - |T| \leq |S^*|$. Putting all together, the total cost of our cutting sequence is less than $(3.5 + \sqrt{2})|S^*|$.

For the case that d and e are to different sides of v, we can also show that the cutting cost is less than $(3.5 + \sqrt{2})|S^*|$. We leave the detail to readers.

Finally, the chord ST can be computed in $O(n + m)$ time, by a clockwise scan of ∂P and a clockwise scan of ∂Q. Other steps also take $O(n + m)$ time. □

2.2 Carving Phase

In this section, we show that if P is enclosed in a minimum axis-aligned rectangle Q, an optimal edge cutting sequence S_e^* is a $(1.5 + \sqrt{2})$-approximation of an optimal cutting sequence S^*. Note that $|Q| \leq \sqrt{2}|P|$ holds in this case.

Theorem 2 *If P is enclosed in a minimum axis-aligned rectangle Q, an optimal edge cutting sequence S_e^* is a $(1.5 + \sqrt{2})$-approximation of an optimal cutting sequence S^*. Moreover, the cutting sequence S_e^* can be found in $O(n^3 + m)$ time.*

Proof. Let us first review the approximation scheme given in [2]. Suppose that an optimal cutting sequence S^* is given. Construct an edge cutting sequence S_e as follows. For every cut $C_v^* \in S^*$, in order, if C_v^* is an edge cut, add it to S_e. Otherwise, C_v^* is tangent to a vertex v of P, and we add to S_e two cuts C_1 and C_2 which are along two edges of P incident at v. Since the original polygon Q is a rectangle, we can select the first cut C_1 such that the part of C_1, not contained in the polygon Q_e obtained after C_2 is made in S_e, is of length at most $|C_v^*|/2$. Hence, the extra cost taken for all such parts is at most $|S^*|/2$ [2].

Let a, b be two endpoints of C_v^* on the boundary of the current polygon Q in S^*, and c, d two endpoints of C_1 and C_2 on the current polygon Q_e ($\subseteq Q$) in S_e. Suppose that a and c are to the same side of v. We now bound the extra cost between $|vc|$ and $|va|$. (The extra cost between $|vd|$ and $|vb|$ can be estimated analogously.) If a and c are on an edge of the original rectangle or some cut preceding C_v^* in S^*, the extra cost between $|vc|$ and $|va|$ is less than $|ac|$. Probably, a is on some optimal cut $C^* \in S^*$ preceding C_v^*, and c is on some edge cut $C_e \in S_e$ preceding C_1. Since c is contained in the polygon Q obtained after $C_v^* \in S^*$ is made, we extend the segment vc until it intersects the cut C^*, say, at the point c'. The extra cost between $|vc|$ ($< |vc'|$) and $|va|$ is less than $|ac'|$. Since all segments representing these costs are disjoint, an upper bound on all extra costs is $|Q| + |S^*|$. Hence, $|S_e| < |S^*| + |S^*|/2 + (|Q| + |S^*|) < 2.5|S^*| + |Q|$.

The extra costs analyzed above are overestimated. The cuts along all edges of P have to appear in S^*, and the segments representing the extra costs described above do not contain any edge of P. The total extra cost can then be reduced by a factor $|P|$. Since $|P| \geq |Q|/\sqrt{2}$, the new upper bound on $|S_e|$ is $(1.5 + \sqrt{2})|S^*|$.

Finally, note that $|S_e^*| \leq |S_e|$ and S_e^* can be found in $O(n^3 + m)$ time. □

Combining Theorem 1 and Theorem 2, we have the following result.

Theorem 3 *Given two convex polygons P and Q, $P \subset Q$, with n and m vertices, respectively, a 7.9-approximation of an optimal cutting sequence for cutting P out of Q can be computed in $O(n^3 + m)$ time.*

Lemma 2 *Given two convex polygons P and Q, $P \subset Q$, with n and m vertices, respectively, an $O(\log n)$-approximation of an optimal cutting sequence can be computed in $O(n + m)$ time.*

Proof. Our separation phase takes $O(n+m)$ time. In the carving phase, the $O(\log n)$-approximation algorithm with $O(n)$ running time in [2] can be used. □

3 Cutting Out Convex Polygons with Ray Cuts

A ray cut originates from infinity and ends at some point within Q. Denote also by S^* an optimal ray cutting sequence. We present below a 6-approximation algorithm for the problem of cutting out convex polygons with ray cuts.

3.1 Separation Phase

Lemma 3 *Let pq be the line segment that gives the closest distance between ∂P and ∂Q. Then, $2|pq| + |P| \leq |S^*|$.*

Proof. Note that all ray cuts in S^* touch the polygon P [2]. Assume that P completely lies in the interior of Q (otherwise, $|pq| = 0$). Then, at least two ray cuts intersect with the original boundary ∂Q, and the cuts along all edges of P have to appear in any ray cutting sequence. Hence, the lemma follows. □

Also, denote by Q' the minimum rectangle containing P, and D_x (resp. D_y) the shortest distance between two vertical edges (resp. horizontal edges) of Q'.

Theorem 4 *There is an $O(n + m)$ time algorithm that cuts out a minimum rectangle containing P, through a ray cutting sequence of cost less than $\frac{5}{\sqrt{2}}|S^*|$.*

Proof. Let pq denote the line segment that gives the closest distance between ∂P and ∂Q, where p is a vertex of P and q is on some edge e of Q. Let l_q denote the supporting line of the edge e, and l'_q the line tangent to P and parallel to l_q such that P is between l_q and l'_q. Let l_{pq} be the line through p and q. See Fig. 2.

Suppose first that l_{pq} is tangent to P at p. Consider a line segment ao tangent to P, with the point a on l_{pq} and the point o on l_q, such that P is between aq and ao. Let b and c be the intersection points of l'_q with ao and aq, respectively. See Fig. 2a. Since $|pq|$ gives the shortest distance between ∂P and ∂Q, $|cp| = D_y$.

Set $H = |cp| + |pq| = D_y + |pq|$ and $Y = |ac|$. It then follows from Lemma 3 that $H \leq |S^*|/2$. We will make two ray cuts along qa and oa. For efficiency, the cost of the cut along qa is considered as two parts $|ac|$ and $|cq|$. Let $C_a = |ac| + |ao|$. Since $|ab| < |ac| + |bc| \leq Y + |bc|$, we have $C_a = |ac| + |aq|(|ab|/|ac|) = Y + (H + Y)((Y + |bc|)/Y) = H + 2Y + (1 + H/Y)|bc|$. Since $|bc| \leq D_x$ holds, consider the function $F(Y) = H + 2Y + (1 + H/Y)D_x$. The minimum value of $F(Y)$ is achieved when $Y = \sqrt{HD_x/2}$. So we have

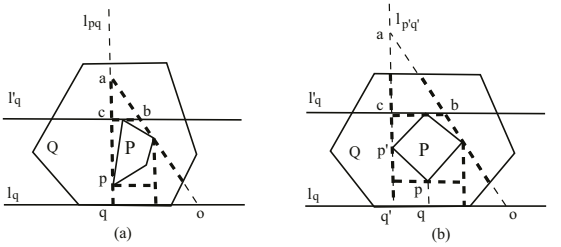


Fig. 2. Illustration of cutting out a minimum rectangle containing P with ray cuts.

$C_{amin} < H + 2\sqrt{2HD_x} + D_x \leq D_x + D_y + |pq| + \sqrt{2}|S^*| \leq \sqrt{2}|P|/2 + |pq| + \sqrt{2}|S^*|$.

To cut out the rectangle Q', we make a ray cut tangent to P and parallel to l_{pq} such that P is between qa and this cut. Finally, make two cuts tangent to P and parallel to l_q. See Fig. 2a. The sum of $|cq|$ and the costs taken for the last three cuts is less than $|Q'| + 2|pq| \leq \sqrt{2}|P| + 2|pq|$. Hence, the total cost is less than $\frac{3}{\sqrt{2}}|P| + 3|pq| + \sqrt{2}|S^*| \leq \frac{3}{\sqrt{2}}(|P| + \sqrt{2}|pq|) + \sqrt{2}|S^*| < \frac{5}{\sqrt{2}}|S^*|$.

Consider now the case that the line l_{pq} divides P into two parts. Let $l_{p'q'}$ denote the line parallel to pq and tangent to P, with $p' \in \partial P$ and $q' \in \partial Q$, and c the intersection point of l'_q with $l_{p'q'}$. See Fig. 2b for an example. Since $|cq'| = D_y + |pq|$, the minimum rectangle containing P can also be cut out through a cutting sequence of cost less than $\frac{5}{\sqrt{2}}|S^*|$.

Since both P and Q are convex, the closest segment pq between ∂P and ∂Q can be found in $O(n + m)$ time. All other steps also take $O(n + m)$ time. □

3.2 Carving Phase

A ray cut is called an *edge-ray* cut if it is made along an edge of the polygon P. Let S_e^* denote an optimal edge-ray cutting sequence. Define the *virtual vertices* of P as the intersection points of the supporting lines of edges of P.

Lemma 4 *Any cut in S_e^* has to end at one of the virtual vertices of P (including the vertices of the polygon P) or on ∂Q before it can reach the target vertex.*

Proof. Omitted in this extended abstract. □

The dynamic programming algorithm for line cuts in [5] is based the observation that after a pair of line cuts C_1 and C_2 are made, the following cuts along the part of ∂P between C_1 and C_2 in clockwise order, are independent from the cuts for the other part of ∂P. It also works for ray cuts, provided that the polygon P is convex. Since an edge-ray cut ends at some virtual vertex of P, two cuts C_1 and C_2 may divide ∂P into at most eight types of configurations. For each of these situations, the subproblems of cutting along two parts of ∂P between C_1 and C_2 can be considered independently.

Theorem 5 *If all cuts are along the edges of the polygon P, then an optimal edge-ray cutting sequence S_e^* can be computed in $O(n^3 + m)$ time.*

Proof. The proof is omitted, as it is similar to that for line cuts [5]. □

Theorem 6 *If P is enclosed in a minimum axis-aligned rectangle Q, then an optimal edge-ray cutting sequence S_e^* is a $(1+\sqrt{2})$-approximation of an optimal ray cutting sequence.*

Proof. Similar to the proof of Theorem 2, we can construct an edge-ray cutting sequence S_e such that $|S_e| < (1+\sqrt{2})|S^*|$ (the detail is omitted). Since $|S_e^*| \leq |S_e|$, the theorem follows. □

Combining Theorems 4, 5 and 6, we have the following result.

Theorem 7 *Given two convex polygons P and Q, $P \subset Q$, with n and m vertices, respectively, a 6-approximation of an optimal ray cutting sequence for cutting P out of Q can be computed in $O(n^3+m)$ time.*

Lemma 5 *Given two convex polygons P and Q, $P \subset Q$, with n and m vertices, respectively, an $O(\log n)$-approximation of an optimal ray cutting sequence can be computed in $O(n+m)$ time.*

Proof. The proof is similar to that of Lemma 2 and thus omitted. *Box*

4 Cutting Out Ray-Cuttable Polygons

In this section, we consider the problem of cutting out a ray-cuttable polygon P out of a convex polygon Q. The difficulty of developing an $O(1)$-approximation algorithm has been shown in [2], as the dynamic programming algorithm does not work for non-convex polygons. Our approach is essentially the same as that in [2], but in the carving phase, we use Theorem 6 to give an $O(\log n)$-approximation.

Let $CH(P)$ denote the convex hull of P. A connected region inside $CH(P)$ but exterior to P is called a *pocket* of P. See Fig. 3. The polygon P can be cut out by first cutting out $CH(P)$ and then all pockects of P from $Q = CH(P)$. Let T denote a pocket of P, and $V(T)$ the set of the vertices of T. Clearly, there is a unique edge e of T such that $e \in Q$ $(= CH(P))$ and $e \notin P$. Let u and v denote two vertices of the edge e (Fig. 3). Denote by SP_{ux} (resp. SP_{vx}), $x \in V(T)$, the shortest path from u (resp. v) to x in T. The region bounded by SP_{ux}, SP_{vx} and the edge uv is called a *funnel*, and denoted by F_x. Both SP_{ux} and SP_{vx} are *inward convex*; they bugle in toward the funnel region. See Fig. 3a for an example, where $x = a$. Clearly, the pocket T is cut out if and only if all funnels $F_x, x \in V(T)$ $(x \neq u$ and $x \neq v)$, are cut out.

The problem of cutting out the pocket T can be reduced to the subproblems of cutting out convex polygons by a recursive procedure [2]. Let $T_{u'v'}$ denote the part of ∂T from u' to v', where u' and v' are two vertices of T such that u' is closer to u on ∂T than v'. First, we find the median vertex a of the vertices of T_{uv}. Let b denote the point of the edge uv, which gives the closest distance of the edge uv to the vertex a in the pocket T. Since P is ray-cuttable, the line segment ab exists. Also, the point b on uv can be found in constant time, provided that

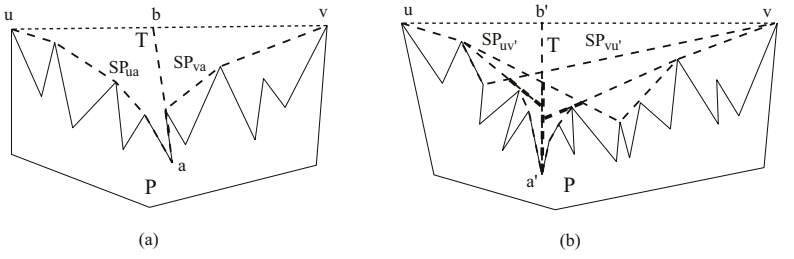


Fig. 3. Illustration for cutting out a ray-cuttable polygon.

SP_{ua} and SP_{va} have been computed. The path SP_{ua} (resp. SP_{va}) is contained in the triangle $\triangle_{uab}$ (resp. $\triangle_{vab}$), which can be considered as the polygon Q. The funnel F_a can thus be cut out by a ray cut along ba and an optimal ray cutting sequence such that all cuts are along the edges of SP_{ua} (resp. SP_{va}) in the triangle $\triangle_{uab}$ (resp. $\triangle_{vab}$). In the next step, we find the median vertex l (resp. r) of T_{ua} (resp. T_{av}), and then cut out the funnel F_l (resp. F_r). In this way, T can be cut out by $O(\log n)$ recursive steps. Let a' be the median vertext of the vertices of some $T_{u'v'}$, and o the interscetion point of two paths $SP_{uv'}$ and $SP_{vu'}$. See Fig. 3b. Since $F_{u'}$ and $F_{v'}$ have been cut out, the part of $F_{a'}$ to be cut out is contained in the region bounded by $u'o$, ov' and $T_{u'v'}$. Since $|u'o| + |ov'| = |T_{u'v'}|$ [2], the total size of the parts of all funnels, which are used in the carving phases, is $O(|P| \log n)$.

Let us now bound the time required to cut out T. All shortest paths SP_{ux} and SP_{vx}, $x \in V(T)$, can be computed in linear time. The time taken for cutting out F_a is $O(n_{uv}^3)$, where n_{uv} is the number of vertices of T_{uv}. Solving $\mathcal{T}(n_{uv}) = 2\mathcal{T}(\frac{n_{uv}}{2}) + O(n_{uv}^3)$ gives the time bound $O(n_{uv}^3)$. All pockets of P can then be cut out in $O(n^3)$ time. Hence, we have the following result (the proof is omitted).

Theorem 8 *Given a ray-cuttable polygon with n vertices drawn on a convex polygon with m vertices, an edge-ray cutting sequence that is an $O(\log n)$-approximation of an optimal ray cutting sequence can be computed in $O(n^3 + m)$ time.*

References

1. J.Bhadury and R.Chandrasekaran, Stock cutting to minimize cutting sequence, *European Journal of Operational Research* **88** (1996) 69-87.
2. O.Daescu and J.Luo, Cutting out polygons with lines and rays, Submtted to IJCGA (see also a preliminary version in *LNCS* **3341** (2004) 669-680).
3. E.D.Demaine, M.L.Demaine and C.S.Kaplan, Polygons cuttable by a circular saw, *Computational Geometry: Theory and Applications* **20** (2001) 69-84.
4. A.Dumitrescu, An approximation algorithm for cuttng out convex polygons, *Computational Geometry: Theory and Applications* **29** (2004) 223-231.
5. M.H.Overmars and E.Welzl, The complexity of cutting paper, *Proc. of the 1st Annual ACM Symposium on Computational Geometry* (1985) 316-321.

Efficient Non-intersection Queries on Aggregated Geometric Data*

Prosenjit Gupta[1], Ravi Janardan[2], and Michiel Smid[3]

[1] International Institute of Information Technology, Gachibowli
Hyderabad 500 019, India
pgupta@iiit.net
[2] Department of Computer Science & Engineering, University of Minnesota
Minneapolis, MN 55455, USA
janardan@cs.umn.edu
[3] Department of Computer Science, Carleton University, Ottawa, Canada K1S 5B6
michiel@scs.carleton.ca

Abstract. Let S be a set of geometric objects that are aggregated into disjoint groups. The problem considered is that of preprocessing S so that for any query object, q, the distinct groups such that no objects from those groups are intersected by q can be reported efficiently. The goal is to devise solutions where the query time is sensitive to the output size, i.e., the number of groups reported. Unfortunately, the obvious approaches of (i) solving the corresponding intersection problem for aggregated data and reporting the complement, or (ii) querying with the complement of q are either expensive or incorrect. Efficient, output-sensitive solutions are given to several non-intersection searching problems on aggregated data, using methods such as geometric duality, sparsification, persistence, filtering search, and pruning.

1 Introduction

Consider the following scenario. The U.S. mutual fund universe consists of about 8,300 mutual funds aggregated into roughly 560 fund families [15][1]. Confronted with this bewildering array of choices, an investor might wish to consolidate his/her holdings by identifying a small number of families whose funds meet desired criteria (e.g., rate of return, volatility, and risk level – each specified as a range of values). This can be accomplished by representing funds as points in $\mathbb{R}^3$, querying them with a box that is the Cartesian product of the three ranges, and sifting through the retrieved funds to identify the associated families. In computational geometry terms, this is merely the *standard* orthogonal range search problem in $\mathbb{R}^3$ [6]. However, this approach is not efficient for the problem

* Research of PG supported, in part, by grant SR/S3/EECE/22/2004 from the Dept. of Science and Technology, Govt. of India. Research of RJ supported, in part, by NSF grant INT–0422775. Research of MS supported, in part, by NSERC.

[1] Worldwide, across 40 major countries, there are about 53,000 funds in an indeterminate number of families [15].

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 544–553, 2005.

at hand since the query time depends on the number of funds satisfying the query range rather than the number of distinct fund families, which can be much smaller. (For instance, a recent query using Yahoo's mutual fund screener yielded 935 funds in just 197 families that had a 1-year return of at least 10% and market capitalization between one and five billion dollars.)

Ideally, one would like a query time that depends on the output size, which here is the number of fund families. This can be done by assigning each point a color based on the family it belongs to and requesting the distinct colors of the points that are in the query range. This is a *generalized* orthogonal range search problem ("generalized" because it includes the standard problem above as a special case, when each color class has cardinality 1). Several types of such generalized problems have been considered recently [1–5, 9–11, 13, 14, 17, 18].

Now, suppose instead that our investor is interested in identifying those fund families for which no funds meet a specified set of criteria. For instance, a conservative investor might want to avoid families whose funds have underperformed with high risk and high volatility. In the generalized setting, the goal would be to report the distinct colors such that no points of those colors are in the query range that specifies the undesirable levels of performance, risk, and volatility.

At first sight, it would appear that this problem can be solved quite easily using the generalized approach, by either (i) reporting the complement of the set of distinct colors found in the range, or (ii) querying with the complement of the query range. Unfortunately, this is not the case. Approach (i) is not output-sensitive since the query time would depend on the number of distinct colors in the range, which can be much greater than the number of colors that avoid the range (the output size). Approach (ii) may not even yield the correct answer: a color found in the complement of the query range could also occur within the range! Furthermore, the complement of the query may not have a compact representation and may need to be split into "simpler" queries (e.g., the complement of a query box would need to be represented as the union of several semi-infinite boxes). Reconciling the responses to these simpler queries and filtering out duplicates could be inefficient. Thus, a different approach is needed.

There are many other examples of such generalized intersection problems. For instance, in VLSI layout design, determining the circuits (i.e., connected components of wires) that are affected/unaffected by the addition of a new wire can formulated as a generalized intersection/non-intersection searching problem.

1.1 Contributions

Formally, our problem is: *Preprocess a set, S, of n colored geometric objects into a data structure and report efficiently, for any query object, q, the distinct colors in S such that q intersects no objects of those colors.*

We call such a problem a *generalized non-intersection problem on S with q*. We measure the efficiency of a solution to this problem by the size and query time of the data structure. Typically, but not exclusively, we seek solutions with linear or close-to-linear storage (e.g., $O(n\text{polylog}(n))$) and low output-sensitive query

Table 1. Summary of results for generalized non-intersection problems on S with q. Rectangles are axes-parallel, t is the number of colors in S, I is the output size, $d > 0$ is an integer constant, and $\varepsilon > 0$ is an arbitrarily small constant.

Underlying space	*Colored objects in S*	*Query q*	*Space*	*Query time*
$\mathbb{R}^1$	points	interval	n	$\log n + I$
$\mathbb{R}^1$	intervals	interval	$n \log^2 n$	$\log n + I \log^2 n$
$\mathbb{R}^2$	points	quadrant	n	$\log n + I$
$\mathbb{R}^2$	points	grounded rectangle	$n \log n$	$\log n + I$
$\mathbb{R}^2$	points	rectangle	n^2	$\log n + I$
$\mathbb{R}^2$	horizontal line segments	vertical line segment	$n \log n$	$\log n + I$
$\mathbb{R}^1$	intervals	point	n	$\log n + I$
$\mathbb{R}^2$	rectangles	point	$n \log^2 n$	$\log n + I \log^2 n$
$\mathbb{R}^d$ $(d \geq 2)$	hyper-rectangles	point	$n^d \log^{d-2} n$	$\log^{d-1} n + I$
$\mathbb{R}^2$	points	halfplane	$n \log n$	$\log^2 n + I$
			tn	$\log n + I$
$\mathbb{R}^3$	points	halfspace	$n \log^2 n$	$n^{1/2+\varepsilon} + I$
			$n^{2+\varepsilon}$	$\log^2 n + I$

times of the form $O(f(n) + I \cdot g(n))$, where f and g are "small" functions (e.g., polylogarithmic or sublinear in n), and I is the output size (i.e., the number of distinct colors reported). Throughout, we make no assumptions about the number of colors in S; it can range from a small constant all the way to n.

To our knowledge, there has, surprisingly, been no systematic study of such generalized non-intersection problems. As mentioned earlier, neither solving the corresponding generalized intersection problem and reporting the complement of the answer nor solving one or more generalized problems with the complement of the query is a satisfactory, or necessarily correct, solution. Here we develop a different set of techniques, based on geometric duality, sparsification, persistence, filtering search, and pruning, to solve several generalized non-intersection problems. Table 1 summarizes our results, some of which are described here. Due to space limitations, we omit all proofs and many details.

2 A General Approach for Non-intersection Queries

We describe a method that can be used for any generalized non-intersection problem on S with q. All that is required is a data structure for the corresponding generalized intersection problem on S with q which can answer counting queries.

We store the distinct colors in S at the leaves of a balanced binary tree, T, in no particular order. For any node v of T, let $C(v)$ be the colors stored in the leaves of v's subtree and let $S(v)$ be the points of S whose colors are in $C(v)$. We store the following information at v: (i) a count, $count(v) = |C(v)|$, of the number of distinct colors in v's subtree; and (ii) a data structure, $G(v)$, that

answers generalized counting queries on $S(v)$ with q; $G(v)$ returns the number of distinct colors among the objects in $S(v)$ that are intersected by q.

To answer a non-intersection query on S with q, we do a depth-first search of T. Let v be the current node and suppose it is a non-leaf. If the count returned by $G(v)$ is $count(v)$, then we abandon the search below v (since q intersects at least one object of $S(v)$ for each color in $C(v)$, the response to the non-intersection query on $S(v)$ with q is the empty set). If the count returned by $G(v)$ is less than $count(v)$, then we search v's subtree recursively (as there is at least one color in $C(v)$ for which q intersects no object in $S(v)$ of that color). If v is a leaf, then we output the color stored at v iff the count returned by $G(v)$ is zero.

This approach is similar to one given by us in [12] for generalized intersection queries, with one key difference. There we pruned the search below a node v iff q did not intersect any object in $S(v)$, which we were able to decide efficiently by using for $G(v)$ a structure that detects if q intersects any object in $S(v)$ – a standard problem. Here, we need to prune the search below v iff q intersects at least one object of each color that occurs among the objects in $S(v)$. This is most easily determined by using a generalized counting structure for $G(v)$, as above; a standard counting would not reveal this information and a standard reporting query would be too expensive.

Theorem 1. *Let $M(m)$ and $f(m)$ be, respectively, the space and query time complexity of the data structure $G(v)$, where $m = |S(v)|$. Assume $M(m)/m$ and $f(m)$ are non-decreasing for non-negative values of m. Then a set, S, of n colored objects can be preprocessed into a data structure of size $\mathcal{S}(n) = O(M(n) \log n)$ so that a generalized non-intersection query on S with q can be answered in time $\mathcal{Q}(n) = O(f(n) + I \cdot f(n) \log n)$, where I is the number of colors reported.*

Applications: The above method can be used to derive efficient solutions for several generalized non-intersection problems, as listed below. (In each case, $G(v)$ is a structure from [9].) We specify these problems below as the ordered 6-tuple $\langle\ S,\ q,\ M(m),\ f(m),\ \mathcal{S}(n),\ \mathcal{Q}(n)\ \rangle$: (i) $\langle$ colored intervals in $\mathbb{R}^1$, interval, $O(m \log m)$, $O(\log m)$, $O(n \log^2 n)$, $O(\log n + I \log^2 n)\ \rangle$; (ii) $\langle$ colored intervals in $\mathbb{R}^1$, point, $O(m)$, $O(\log m)$, $O(n \log n)$, $O(\log n + I \log^2 n)\ \rangle$; (iii) $\langle$ colored points in $\mathbb{R}^2$, axes-parallel rectangle, $O(m^2 \log^2 m)$, $O(\log^2 m)$, $O(n^2 \log^3 n)$, $O(\log^2 n + I \log^3 n)\ \rangle$; and (iv) $\langle$ axes-parallel rectangles in $\mathbb{R}^2$, point, $O(m \log m)$, $O(\log m)$, $O(n \log^2 n)$, $O(\log n + I \log^2 n)\ \rangle$.

3 Querying Points with Orthogonal Ranges

3.1 Querying with Intervals in $\mathbb{R}^1$

Given a set, S, of n colored points on the real line, $\mathbb{R}^1$, we wish to answer a generalized non-intersection query with an interval $q = [a, b]$.

For each color, c, let S_c be the points of color c. We sort the points in S_c in nondecreasing order as $p_1, p_2, \ldots, p_k$, where $k = |S_c|$. We transform this sequence of points into a sequence of intervals $(-\infty, p_1], [p_1, p_2], \ldots [p_k, +\infty)$, which we denote by L_c.

Lemma 1. *Query interval $q = [a, b]$ does not contain any points of color c iff q is contained properly in exactly one interval of L_c.*

Lemma 1 implies that the original generalized non-intersection problem can be re-phrased as the standard problem of reporting all intervals that contain a query interval. To solve this, we map each input interval $[x, y]$ to a point (x, y) in $\mathbb{R}^2$, and associate with it the color of $[x, y]$. Note that $q = [a, b] \subset [x, y]$ iff $x < a$ and $y > b$, i.e. iff point (x, y) is contained properly in the northwest quadrant of the point (a, b). The points contained in such a query quadrant can be reported efficiently by storing the points (x, y) in a priority search tree [16]. The solution can be made dynamic, to accommodate the insertion and deletion of colored points in $\mathbb{R}^1$. For each color c, we maintain the points of S_c in a balanced binary search tree. We use this to update L_c whenever a point is inserted or deleted in S_c, and then update the priority search tree with the corresponding point in $\mathbb{R}^2$, all in time $O(\log n)$. (We note that the static problem on integer inputs has been solved independently in [17], in time $O(I)$, using a different approach.)

Theorem 2. *A set, S, of n colored points on $\mathbb{R}^1$ can be preprocessed into a data structure of size $O(n)$, so that for any query interval q, a generalized non-intersection query on S with q can be answered in time $O(\log n + I)$. Moreover, colored points can be inserted or deleted in S in time $O(\log n)$.*

3.2 Querying with Quadrants in $\mathbb{R}^2$

Given a set, S, of n colored points in $\mathbb{R}^2$, we wish to answer a generalized non-intersection query for a quadrant defined by a query point $q = (a, b)$. Specifically, we consider the north-east quadrant, $NE(q)$, defined as the set of all points (x, y) such that $x \geq a$ and $y \geq b$. (Note that this is different from the quadrant problem considered in Section 3.1, which is a standard intersection problem.)

As before, let S_c be the set of points of color c. Let M_c be the set of *maximal points* of S_c; that is, $M_c \subseteq S_c$ consists of points each of whose north-east quadrants contains no point of S_c.

M_c can be used to define a set, H_c, of horizontal line segments that form a staircase-like structure which descends rightwards. Specifically, sort M_c by decreasing y-coordinates. For each pair of consecutive points, (x_p, y_p) and (x_r, y_r), in this order, where $y_p > y_r$ (note that $x_p < x_r$), insert the horizontal segment $((x_p, y_r), (x_r, y_r)]$ into H_c. Also include in H_c, a horizontal ray, from (x_h, y_h) to $(-\infty, y_h)$, where (x_h, y_h) is the highest point of M_c, and a horizontal ray from $(x_\ell, -\infty)$ to $(\infty, -\infty)$, where (x_ℓ, y_ℓ) is the lowest point of M_c. Note that all the segments in H_c (including the two rays) are open on the left.

Intuitively, if $NE(q)$ does not contain any point of S_c, hence also of M_c, then it does not intersect any segment from H_c. Thus, q must lie "above" the staircase defined by H_c. All such staircases can be computed by shooting a vertical ray, $Ray(q)$ downwards from q. In what follows, the intersection of $Ray(q)$ with a segment is said to be proper if q is strictly above the segment.

Lemma 2. *Let $Ray(q)$ be the ray that emanates from q and is directed downwards. Then $NE(q)$ contains no point of S_c iff $Ray(q)$ intersects properly exactly one segment of H_c.*

Our goal now is to solve the standard problem of reporting the segments of H_c, for all colors c, that are intersected by $Ray(q)$. Given S, we compute and store the segments of H_c, for all colors c, in a hive-graph [6] and query this with $Ray(q)$, for any query point q.

Theorem 3. *A set, S, of n colored points in the plane can be preprocessed in time $O(n \log n)$ into a data structure of size $O(n)$, so that for any query point q, a generalized non-intersection query on S with the north-east quadrant of q can be answered in time $O(\log n + I)$.*

In the full paper, we also show how to semi-dynamize the above result using $O(n)$ space and $O(\log^2 n + I)$ query time, with an insertion time of $O(\log n)$ when amortized over n successive insertions into an empty set.

3.3 Querying with Grounded Rectangles in $\mathbb{R}^2$

The data structure underlying Theorem 2 can be coupled with the notion of persistence [7] to answer generalized non-intersection queries on a set, S, of n colored points in $\mathbb{R}^2$ with a grounded query rectangle $q = [a, b] \times [f, \infty)$.

We create a linked list, L, which contains the c-colored point of maximum y-coordinate, for each color c (ties broken arbitrarily). Next, we sort the points of S by non-increasing y-coordinates and insert them in this order into a partially persistent version of the structure of Theorem 2, using the x-coordinate as the key. To answer a query $q = [a, b] \times [f, \infty)$, we access the persistent version corresponding to the smallest y-coordinate greater than or equal to f and query it with the interval $[a, b]$. Additionally, we traverse L in the order of nondecreasing y-coordinates and report all colors with maximum y-coordinate less than f.

Note that it is possible that there are colors such that all points of those colors have y-coordinate less than f. These colors need to be reported but they will not be found when querying the partially persistent structure; hence the need for list L.

Theorem 4. *A set, S, of n colored points in $\mathbb{R}^2$ can be stored in a structure of size $O(n \log n)$, so that for any query rectangle $q = [a, b] \times [f, \infty)$, a generalized non-intersection query on S with q can be answered in $O(\log n + I)$ time.*

3.4 Querying with Orthogonal Rectangles in $\mathbb{R}^2$

The data structure of Theorem 4 can be incorporated within the filtering search paradigm [6] to answer generalized non-intersection queries on a set S of n colored points in $\mathbb{R}^2$ with a general (axes-parallel) query rectangle $q = [a, b] \times [f, g]$. The idea is to partition $\mathbb{R}^2$ into regions within which q behaves like a grounded rectangle. However, in doing so, q may retrieve colors that are not

in the true output. Fortunately, the number of such overreported colors will be small enough that they can be filtered out efficiently.

Let $p_1, p_2, \ldots, p_n$ be the points of S, sorted in increasing order of their y-coordinates. For each $1 \leq k \leq n/\log n$, let $S_k = \{p_1, p_2, \ldots, p_{k \log n}\}$, and let y_k be the y-coordinate of the point $p_{k \log n}$. We store each set S_k in an instance, D_k, of the data structure of Theorem 4.

For every color, we find the point of that color in S which has minimum y-coordinate and store all these points in a list L, sorted in decreasing order of their y-coordinates. We initialize to zero an integer array B whose length is equal to the number of distinct colors that occur in S.

A query on S, with $q = [a, b] \times [f, g]$, is answered as follows:

1. Using binary search, find the index k such that $y_k \leq g < y_{k+1}$. (Thus, q's upper edge is above the line $y = y_k$. W.r.t. the subset of S on or below this line (i.e., S_k), q functions like the grounded rectangle $q' = [a, b] \times [f, \infty)$.)
2. Query D_k with q', and store the colors found to not be in q' in a list A. For each color c in A, set $B[c] = 1$, and store with $B[c]$ a pointer to its occurrence in A. (Note that, at this stage, A may contain colors that are actually in $q = [a, b] \times [f, g]$. It may also not contain certain colors that are not in q because these colors do not occur in S_k. The overreported colors are filtered out and the underreported colors are discovered in the next few steps.)
3. For each j, where $k \log n + 1 \leq j \leq (k+1) \log n$, if p_j is in q and if $B[c] = 1$, where c is the color of p_j, then set $B[c] = 0$, follow the pointer that is stored with $B[c]$ to the occurrence of c in the list A, and delete c from A. (This removes from A those colors that were overreported in step 2.) On the other hand, if p_j is not in q and $B[c] = 0$, then add c to A. (This includes in A some of the colors that were underreported in step 2.)
4. Scan L, by decreasing y-coordinates, and find the colors of all points with y-coordinate larger than y_{k+1}. For each of these colors c, set $B[c] = 1$ and add c to the list A. (This includes in A the remainder of the colors that were underreported in step 2. Such colors will be discovered by scanning L since it stores for each color the point with minimum y-coordinate.)
5. For each color c in A set $B[c] = 0$ and return A as the answer to q.

Theorem 5. *A set, S, of n colored points in $\mathbb{R}^2$ can be preprocessed into a data structure of size $O(n^2)$, so that for any query rectangle $q = [a, b] \times [f, g]$, a non-intersection query on S with q can be answered in time $O(\log n + I)$.*

4 Queries Involving Orthogonal Line Segments

We consider generalized non-intersection queries on colored horizontal line segments in $\mathbb{R}^2$ with a vertical query segment q, whose endpoints are (a, f) and (a, g). Our solution uses a persistent version of the structure of Theorem 2 and also a persistent red-black tree.

Call the distinct colors of the segments that intersect the supporting line of q the *active colors*; the remaining colors are the *inactive colors*. We can solve

our generalized non-intersection problem, w.r.t. the active colors, by using an instance of the data structure in Theorem 2. We make the structure of Theorem 2 persistent, and query the appropriate version with q. Note that this only reports the active colors that are not intersected by q. We also need to report all the inactive colors. For this, we track the set of inactive colors over all possible queries q, again using persistence.

We sort the endpoints of the segments in S by nondecreasing x-coordinates (favoring right endpoints over left endpoints, in the case of ties), and sweep over them with a vertical line L. Let D be the data structure of Theorem 2, T be a red-black tree, and let B be an array indexed by color. At any time in the sweep D stores a set of colored points that are the y-coordinates of the segments that are intersected currently by L, T stores the inactive colors, and B stores, for each color, the number of segments of that color that are intersected by L.

Initially, L is to the left of all the segments, so no segments are intersected by it and all the colors are inactive. Thus, D is empty, T contains all the colors, and B is all-zero. We build the persistent versions of D and T as follows.

In a general step, suppose that L reaches an endpoint e of a horizontal segment h, of color c. If e is the left endpoint of h, then we insert persistently the y-coordinate of h in D, with color c. We increment $B[c]$ and, if $B[c] = 1$ now, then we delete c persistently from T. If e is the right endpoint of h, then we delete persistently the y-coordinate of h from D. We decrement $B[c]$ and, if $B[c] = 0$ now, then we insert c persistently into T.

Denote the persistent versions of D and T as $\mathcal{D}$ and $\mathcal{T}$, respectively. Given q, with endpoints (a, f) and (a, g), we locate in $\mathcal{D}$ the instance of D corresponding to the largest x-coordinate that is at most a and answer the query underlying Theorem 2 using the interval $[f, g]$. In addition, we also output the colors in the corresponding instance, T, of $\mathcal{T}$.

Theorem 6. *A set, S, of n colored, horizontal line segments in $\mathbb{R}^2$ can be preprocessed in time and space $O(n \log n)$, so that for any vertical query segment, q, a generalized non-intersection query on S with q can be answered in time $O(\log n + I)$.*

5 Querying Points with Halfspaces in $\mathbb{R}^d$

We consider how to answer generalized non-intersection queries on a set, S, of n colored points in $\mathbb{R}^d$ with a query halfspace bounded by a hyperplane q. Specifically, if x_i, $1 \leq i \leq d$, are the coordinate axes, then the query is the open halfspace, q^-, consisting of points that lie below q in direction x_d.

W.l.o.g. assume that q is non-vertical (the vertical case is easy). Our approach is based on transforming the problem to a standard intersection problem in a dual space. Let $\mathcal{F}$ denote the well-known point-hyperplane duality transform [8]: If $p = (p_1, \ldots, p_d)$ is a point in $\mathbb{R}^d$, then $\mathcal{F}(p)$ is the hyperplane $x_d = 2p_1x_1 + \cdots + 2p_{d-1}x_{d-1} - p_d$. If $H : x_d = a_1x_1 + \cdots + a_{d-1}x_{d-1} + a_d$ is a non-vertical hyperplane in $\mathbb{R}^d$, then $\mathcal{F}(H)$ is the point $(a_1/2, \ldots, a_{d-1}/2, -a_d)$.

Using $\mathcal{F}$ we map the colored points in S to a set, S', of colored hyperplanes and map q to the point $q' = \mathcal{F}(q)$, all in $\mathbb{R}^d$. The halfspace q^+ is mapped to a ray, $Ray(q)$, which emanates from q' and is directed downwards.

Lemma 3. *Let E_c be the upper-envelope of the hyperplanes of color c. There are no points of color c in q^- iff $Ray(q)$ intersects E_c. Moreover, if $Ray(q)$ intersects E_c, then it does so exactly once.*

It suffices to determine from the collection of envelopes of different colors, those that are intersected by $Ray(q')$; the number, k, of such intersected envelopes is also the number, I, of colors not in q^-. Efficient solutions are known for this ray-envelope intersection problem [10] in $\mathbb{R}^2$ and $\mathbb{R}^3$.

Theorem 7. *For a set, S, of n colored points in $\mathbb{R}^d$ and any query hyperplane q, a generalized non-intersection query on S with an open halfspace of q can be answered in $O(n \log n)$ space and $O(\log^2 n + I)$ query time for $d = 2$, and $O(n \log^2 n)$ (resp. $O(n^{2+\varepsilon})$) space and $O(n^{1/2+\varepsilon} + I)$ (resp. $O(\log^2 n + I)$) query time for $d = 3$.*

5.1 A More Efficient Solution in $\mathbb{R}^2$ for Few Colors

The above solution for $d = 2$ can be improved when t – the number of colors in S – is $O(\log n)$. We represent each set of colored points by a sparse subset consisting of their convex hull and establish a necessary and sufficient condition for a color to be reported. Then, using geometric duality, we transform the generalized non-intersection problem to a standard one and solve the latter.

For each color c, let CH_c denote the convex hull of the points of color c. The boundary of CH_c can be partitioned into an upper chain, U_c, and a lower chain, L_c, whose endpoints are the leftmost and rightmost vertices of CH_c.

Lemma 4. *No c-colored point is in q^- iff L_c is in the closed halfplane q^+.*

Our goal now is to report the distinct colors, c, such that L_c is in q^+. We map this to a standard intersection problem in the dual space, via $\mathcal{F}$. Thus, L_c maps to an infinite convex chain L'_c which is convex downwards, q maps to a point q', and q^+ maps to a vertical ray, $Ray(q)$, emanating downwards from q'. Moreover, L_c is in q^+ iff $Ray(q)$ intersects L'_c. Also, by convexity, if $Ray(q)$ intersects L'_c, then it does so exactly once. Thus, our problem can be solved by simply reporting the dual chains intersected by $Ray(q)$.

We associate with each dual chain, L_c, the color c. We draw a vertical line through each vertex and intersection point in the set of dual chains of the different colors. Within each strip so obtained, the chain segments are non-intersecting, so they can be totally ordered. Given $Ray(q)$, we locate, via binary search, the strip containing the x-coordinate of q', and then do a second binary search within this strip using the y-coordinate of q'. We report the color of each chain for which there is a segment in the strip that is on or below q'.

Theorem 8. *A set, S, of n colored points in $\mathbb{R}^2$, drawn from a palette of t colors, can be stored in a structure of size $O(tn)$, so that for any query line, q, a generalized non-intersection query on S with an open halfplane of q can be answered in time $O(\log n + I)$.*

References

1. P.K. Agarwal, S. Govindarajan, S. Muthukrishnan. Range Searching in categorical data: colored range searching on grid-trees, *Proc. 10th European Symp. on Algorithms*, LNCS 2461, Springer 2002, 323–334.
2. P.K. Agarwal and M. van Kreveld. Connected component and simple polygon intersection searching. *Algorithmica* 15(6), 1996, 626–660.
3. P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. New upper bounds for generalized intersection searching problems. *Proc. 22nd ICALP*, LNCS 944, Springer 1995, 464–475.
4. P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. Red-Blue intersection reporting for objects of non-constant size. *The Computer J.*, 39(6), 1996, 541–546.
5. P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. New results on intersection query problems. *The Computer J.*, 40(1), 1997, 22–29.
6. B. Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Computing*, 15(3), 1986, 703–724.
7. J.R. Driscoll, N. Sarnak, D.D. Sleator, and R.E. Tarjan. Making data structures persistent. *J. Computer and System Sciences*, 38:86–124, 1989.
8. H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag (1987).
9. P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization, *J. Algorithms*, 19, 1995, 282–317.
10. P. Gupta, R. Janardan, and M. Smid. Algorithms for generalized halfspace range searching and other intersection searching problems, *Computational Geometry: Theory and Applications*, 5, 1996, 321–340.
11. P. Gupta, R. Janardan, and M. Smid. A technique for adding range restrictions to generalized searching problems. *Information Processing Letters*, 64, 1997, 263–269.
12. P. Gupta, R. Janardan and M. Smid. Algorithms for some intersection searching problems involving circular objects, *Intl. J. Math. Algorithms*, 1, 1999, 35–52.
13. P. Gupta, R. Janardan and M. Smid. Computational Geometry: Generalized Intersection Searching *Handbook of Data Structures*, S. Sahni and D. Mehta, eds., CRC Press, 2004, Ch. 64, 1–17.
14. R. Janardan and M. Lopez. Generalized intersection searching problems. *Intl. J. Computational Geometry & Applications*, 3, 1993, 39–69.
15. Information culled from the Investment Company Institute's Mutual Fund Factbook, 2003 (`www.ici.org`), and Yahoo.com (`finance.yahoo.com`).
16. E.M. McCreight. Priority search trees, *SIAM J. Computing*, 14(2), 1985, 257–276.
17. S. Muthukrishnan. Efficient algorithms for document retrieval problems. *Proc. 13th Annual Symp. on Discrete Algorithms*, 2002, 657–666.
18. Q. Shi and J. JaJa. Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines. Technical Report CS–TR–4542, UMIACS Univ. of Maryland, College Park, MD, 2003.

An Upper Bound on the Number of Rectangulations of a Point Set*

Eyal Ackerman, Gill Barequet, and Ron Y. Pinter

Dept. of Computer Science
Technion–Israel Institute of Technology, Haifa 32000, Israel
{ackerman,barequet,pinter}@cs.technion.ac.il

Abstract. We consider the number of different ways to divide a rectangle containing n noncorectilinear points into smaller rectangles by n non-intersecting axis-parallel segments, such that every point is on a segment. Using a novel counting technique of Santos and Seidel [12], we show an upper bound of $O(20^n/n^4)$ on this number.

1 Introduction

Given a set P of n points within an axis-parallel rectangle R, a *rectangulation* of (R, P) is a set of non-intersecting segments that partitions R into smaller rectangles, such that every point in P is on a segment. See Figure 1 for examples of rectangulations.

The problem of finding a rectangulation with a minimum total length of the segments has attracted considerable attention in the literature. Lingas, Pinter, Rivest, and Shamir [10] introduced it as a special case of a problem with applications to VLSI design, and showed that it is NP-hard. Since then, several approximation algorithms have been suggested (e.g., [7–9]), including a polynomial-time approximation scheme [11]. De Meneses and de Souza [6] suggested integer-programming formulations and techniques to find exact solutions for medium sized instances of the problem.

When the points are *noncorectilinear*, i.e., no two points share the same x or y coordinate, the complexity class of the minimization problem is unknown. However, it can be shown [3] that the optimal solution in this case consists of exactly n segments. Hereafter, we consider only such rectangulations and investigate the following question:

> Given a set P of n noncorectilinear points within a rectangle R, how many different rectangulations (by n segments) of (R, P) are there?

A similar question, that of the number of *triangulations* of the convex hull of a set of n points in the plane, has attracted considerable attention in the

* Work on this paper by the first author has been supported in part by a Neeman fellowship at the Technion. Work by the first and second authors has also been supported in part by the European FP6 Network of Excellence Grant 506766 (AIM@SHAPE).

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 554–559, 2005.

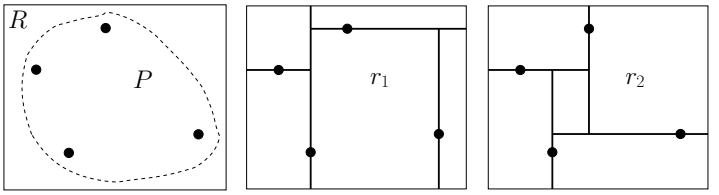


Fig. 1. Rectangulations of (R, P)

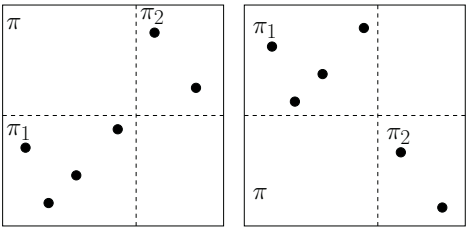


Fig. 2. Point sets in separable permutations

literature. The first singly-exponential upper bound on the number of triangulations, $O(173,000^n)$, was given by Smith [15]. The upper bound was improved by Seidel [13] to $O(2^{12.245113n-\Theta(\log n)}) \approx O(4,855^n)$ and by Denny and Sohler [5] to $O(2^{8.2n+O(\log n)}) \approx O(294^n)$. The best currently-known upper bound, $O(59^n/n^6)$, is due to Santos and Seidel [12].

In a previous paper [1] we observed that the number of rectangulations of a point set P depends only on the relative order of the points in P, which can be represented by a permutation on n. We proved that if the permutation of the points is *separable* [2][1], then the number of rectangulations is exactly the $(n+1)$st Baxter number, which is [4, 14]:

$$\mathrm{B}(n+1) = \sum_{r=0}^{n} \frac{\binom{n+2}{r}\binom{n+2}{r+1}\binom{n+2}{r+2}}{\binom{n+2}{1}\binom{n+2}{2}} = \Theta(8^n/n^4).$$

(In [1] we also observed that the number of separable permutations on n is the $(n-1)$st Schröder number $r_n = \sum_{k=0}^{n} 2^k \binom{n}{k}\binom{n}{k-1}/n = \Theta((3+\sqrt{8})^n/n^{1.5})$. Thus,

[1] A separable permutation is either a permutation on one element or the concatenation of two separable permutations. Formally, let $\pi_1 = (\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_n)$ and $\pi_2 = (\beta_1, \beta_2, \beta_3, \ldots, \beta_m)$ be two permutations on n and m, respectively. We say that $\pi = (\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{n+m})$ is the result of concatenating π_2 *above* π_1 if $\sigma_i = \alpha_i$ for $1 \leqslant i \leqslant n$ and $\sigma_{n+i} = n + \beta_i$ for $1 \leqslant i \leqslant m$ (see Figure 2(a)). Likewise, we say that $\pi = (\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{n+m})$ is the result of concatenating π_2 *below* π_1 if $\sigma_i = m + \alpha_i$ for $1 \leqslant i \leqslant n$ and $\sigma_{n+i} = \beta_i$ for $1 \leqslant i \leqslant m$. (see Figure 2(b)). Then, a permutation π is a *separable* if 1. $\pi = (1)$; or 2. There are two separable permutations π_1 and π_2 such that π is the the concatenation of π_2 above or below π_1.

the portion of separble permutations out of the $n!$ permutations is asymptotically zero.)

We observed empirically that the number of rectangulations of all other sets of n points in non-separable permutations is strictly larger than the $(n+1)$st Baxter number. This was done by counting systematically all the rectangulations of sets of up to 10 points in all possible permutations. Nevertheless, we were unable to prove that this is true, that is, that the $(n+1)$st Baxter number is a lower bound on the number of rectangulations of all point sets of size n.

It is easy to show super-exponential upper bounds. For example, assume (without loss of generality) that there are fewer vertical segments than horizontal segments in any rectangulation. Then choose the endpoints of the at most $n/2$ vertical segments; for each such segment there are no more than $\binom{n+1}{2}$ options. After determining the vertical segments, all the horizontal segments are unique: they extend on both sides of the yet unused points until hitting the *interior* of the first vertical segment (or the bounding rectangle). This yields the upper bound $O(\binom{n+1}{2}^{n/2})$, which is $O(n^n)$.

Another method uses the fact that the number of "point-free" rectangulations (also known as *floor-plans* – subdivisions of a rectangle into smaller isothetic rectangles) is also related to Baxter numbers [16] and is thus $\Theta(8^n/n^4)$. Each such floor-plan can be trivially associated with at most $n!$ permutations, hence we obtain the slightly upper bound $O(n!\,8^n/n^4)$.

An even better – but still super-exponential – upper bound can be obtained from the fact that in any rectangulation there always exists a segment s that touches at most three other segments. By removing s and the point p on it (and extending the segment supported by s, if such segment exists), we obtain a rectangulation of $n-1$ points. Now, there are exactly six possible ways of inserting s into this rectangulation. Suppose s was horizontal. Then, if s touched exactly two vertical segments, we stretch a horizontal segment from p, in both directions, until hitting a vertical segment. If s touched exactly three vertical segments, then there are two possibilities: s must "chop" the first vertical segment either to its right or to its left. Since s might be vertical, we have a total of six possibilities. To be able to construct the rectangulation all we need to store is the way every point and segment are added and the order of the points. Thus the number of rectangulations is $O(n!\,6^n)$.

In the following section we show that the number of rectangulations of a set of n noncorectilinear points (arranged in any arbitrary permutation) is $O(20^n/n^4)$. This is the first proven singly-exponential upper bound on the number of rectangulations of any point set.

2 The Upper Bound

Our main result is:

Theorem 1. *The maximum number of rectangulations of n noncorectilinear points (by n segments) is $O(20^n/n^4)$.*

Proof. The proof follows the structure of the proof of the upper bound on the number of triangulations of a planar point set, given in [12]. We denote by $f(n)$ the maximum number of rectangulations of n points. Let P be a set of n noncorectilinear points within a rectangle R, and let r be a rectangulation of (R, P). A *T-junction* is an endpoint of a segment on another segment, or on the boundary. The *degree* of a point $p \in P$ in r is the number of T-junctions on the segment that contains p. For example, the rightmost point in P in Figure 1 has degree 2 in r_1 and degree 3 in r_2. Let n_i^r be the number of points with degree i in r, then clearly $n = \sum_i n_i^r$.

Every segment is bounded by two T-junctions, thus every segment s contributes at most four to the total sum of degrees: two to the point it contains, and one to every point that is contained in a segment bounding s (if it is not a boundary segment). Note that the point on s might have a degree greater than four, however we charge other segments for their contribution to this degree. Therefore, the total sum of degrees is $4n - b$, where b is the number of T-junctions on the boundary of R in r. It is easy to verify that if $n \geq 3$, then $b \geq 4$. Thus, for $n \geq 3$ we have

$$4n - 4 \geq \sum_i i \cdot n_i^r.$$

Easy manipulations show that

$$4 \sum_i n_i^r \geq 4 + \sum_i i \cdot n_i^r,$$

$$\sum_i (4 - i) n_i^r \geq 4, \text{ and}$$

$$\sum_i (5 - i) n_i^r \geq 4 + \sum_i n_i^r = n + 4.$$

Considering only the positive summands on the left-hand side of the last equation we have:

$$3n_2^r + 2n_3^r + n_4^r \geq n + 4. \tag{1}$$

Denote by h_i the maximum number of rectangulations of (R, P) that one can obtain by adding some point $p \in P$ to a rectangulation r' of $(R, P \setminus \{p\})$ and "stretching" the segment through p such that the degree of p in the resulting rectangulation is i. Clearly, $h_2 = 2$, since the segment through p can be either vertical or horizontal and we must stop "stretching" it as soon as it hits another segment in each direction. Similarly, $h_3 = 4$, since when the orientation of the segment through p is horizontal (resp., vertical), then we must "chop" the first segment either to the left (resp., below) or to the right (resp., above) of p. Note that segments that were supported by the chopped part of the segment are extended until they hit another segment or the boundary (see Figures 3(d,e) for examples). Likewise, $h_4 \leq 6$ and in general $h_i \leq 2(i - 1)$.

Let N_i be the number of points with degree i in all the rectangulations of (R, P). Then,

$$N_i \leq n \cdot h_i \cdot f(n - 1),$$

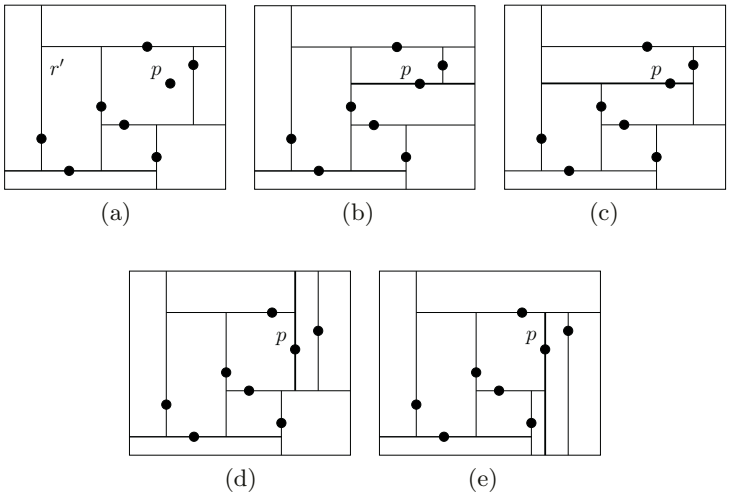


Fig. 3. Four possible ways of adding p to r' such that the degree of p is 3

Table 1. Empirical results of the maximum number of rectangulations

n	B$(n+1)$	Maximum number of rectangulations
4	92	93
5	422	428
6	2,074	2,122
7	10,754	11,092
8	58,202	60,524
9	326,240	342,938
10	1,882,960	2,000,856

and specifically $N_2 \leq 2n \cdot f(n-1)$, $N_3 \leq 4n \cdot f(n-1)$, and $N_4 \leq 6n \cdot f(n-1)$.

We now prove by induction on n that $f(n) \leq 20^n/\binom{n+4}{4}$. For $n = 0, 1, 2$ the claim holds trivially ($f(0) = 1 = 20^0/\binom{4}{4}$, $f(1) = 2 < 4 = 20^1/\binom{5}{4}$, and $f(2) = 6 < 26.666... = 20^2/\binom{6}{4}$). Now assume that the claim holds for all $n' < n$, for $n \geq 3$. By summing Equation 1 over all possible rectangulations, we have:

$$3N_2 + 2N_3 + N_4 \geq (n+4)f(n) \tag{2}$$

On the left-hand side of Equation 2 we have:

$$20n \cdot f(n-1) \leq 20n\frac{20^{n-1}}{\binom{n+3}{4}} = (n+4)\frac{20^n}{\binom{n+4}{4}}.$$

Hence $f(n) = O(20^n/n^4)$, and the claim follows.

3 Conclusions

We have showed that the number of rectangulations of a set of n noncorectilinear points is $O(20^n/n^4)$. However, according to our experiments for small values of n (see Table 1), it seems that the maximum number of rectangulations is much closer to the $B(n+1) = \Theta(8^n/n^4)$ lower bound from [1]. As mentioned in the introduction, we also believe that for *every* set of n (noncorectilinear) points, the number of rectangulations is at least the $(n+1)$st Baxter number.

References

1. E. Ackerman, G. Barequet, and R.Y. Pinter, On the number of rectangular partitions, *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms*, New Orleans, LA, January 2004, 729–738.
2. P. Bose, J.F. Buss, and A. Lubiw, Pattern matching for permutations, *Information Processing Letters*, 65 (1998), 277–283.
3. F.C. Calheiros, A. Lucena, and C.C. de Souza, Optimal rectangular partitions, *Networks*, 41:1 (2003), 51–67.
4. F.R.K. Chung, R.L. Graham, V.E. Hoggatt, and M. Kleiman, The number of Baxter permutations, *J. Combinatorial Theory, Ser. A*, 24 (1978), 382–394.
5. M. Denny and C. Sohler, Encoding a triangulation as a permutation of its point set, *Proc. 9th Canadian Conf. on Compututational Geometry*, Kingston, Ontario, Canada, August 1997, 39–43.
6. C.N. de Meneses and C.C. de Souza, Exact solutions of rectangular partitions via integer programming, *Int. J. of Computational Geometry and Applications*, 10 (2000), 477–522.
7. D.Z. Du, L.Q. Pan, and M.T. Shing, Minimum edge length guillotine rectangular partition, Technical Report MSRI 02418-86, University of California, Berkeley, CA, 1986.
8. T.F. Gonzalez and S.-Q. Zheng, Improved bounds for rectangular and guillotine partitions, *J. of Symbolic Computation*, 7 (1989), 591–610.
9. T.F. Gonzalez and S.-Q. Zheng, Approximation algorithms for partitioning a rectangle with interior points, *Algorithmica*, 5 (1990), 11–42.
10. A. Lingas, R.Y. Pinter, R.L. Rivest, and A. Shamir, Minimum edge length rectilinear decompositions of rectilinear figures, *Proc. 20th Allerton Conf. on Communication, Control, and Computing*, Monticello, IL, 1982, 53–63.
11. J.S.B. Mitchell, Guillotine subdivisions: Part II – A simple polynomial-time approximation scheme for geometric k-MST, TSP, and related problems, *SIAM J. on Computing*, 28 (1999), 1298–1309.
12. F. Santos and R. Seidel, A better upper bound on the number of triangulations of a planar point set, *J. Combinatorial Theory, Ser. A*, 102 (2003), 186–193.
13. R. Seidel, On the number of triangulations of planar points sets, *Combinatorica*, 18 (1998), 297–299.
14. Z.C. Shen and C.C.N. Chu, Bounds on the number of slicing, mosaic, and general floorplans, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22:10 (2003), 1354–1361.
15. W.D. Smith, Studies in computational geometry motivated by mesh generation, *Ph.D. Thesis*, Princeton University, 1989.
16. B. Yao, H. Chen, C.K. Cheng, and R. Graham, Floorplan representations: Complexity and connections, *ACM Trans. on Design Automation of Electronic Systems*, 8 (2003), 55–80.

Opportunistic Data Structures for Range Queries*

Chung Keung Poon and Wai Keung Yiu

Department of Computer Science, City University of Hong Kong
{ckpoon,kwkyiu}@cs.cityu.edu.hk

Abstract. In this paper, we study the problem of supporting range sum queries on a compressed sequence of values. For a sequence of n k-bit integers, $k \leq O(\log n)$, our data structures require asymptotically the same amount of storage as the compressed sequence if compressed using the Lempel-Ziv algorithm. The basic structure supports range sum queries in $O(\log n)$ time. With an increase by a constant factor in the storage complexity, the query time can be improved to $O(\frac{\log \log n}{\log \log \log n} + k)$.

1 Introduction

With the proliferation of electronic data nowadays, there is a growing demand to store data in compressed form. In fact, data compression has long been recognized as an important area in computer science and engineering. Numerous compression algorithms [3, 7, 10, 15] have been devised and used in day-to-day computer operations to reduce the storage requirement for data as well as the necessary bandwidth for data transmission.

Very often, we need to retrieve or operate on only part of the data. It would be desirable if the data is compressed in such a way that one could inspect or extract information about part of the original data directly from the compressed data. For example, textual information can be stored and indexed by a compressed suffix array [8, 13, 14] or Burrows-Wheeler Transform with Lempel-Ziv algorithm [5] so that one can search for a pattern in the text efficiently. However, for many fundamental data structural problems including range sum queries, we are not aware of any focused research reported in the literature. Thus we are motivated to study the range sum query problems in this paper.

Formally, our problem can be defined as follows. Given an array $A[0..n-1]$ of k-bit integers, $k \leq O(\log n)$, compress it so that range sum queries, i.e., the sum of values in $A[i..j]$ given the boundaries i and j, can be answered efficiently. Note that when $i = j$, the range query becomes a point query. That means, the data structure should be able to report every entry of A without decompressing the whole data. Clearly, the major performance measures of interest are the storage and query complexities. We will assume a unit-cost word RAM with word size

* The work described in this paper is fully supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (CityU 1071/02E).

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 560–569, 2005.

$\Theta(\log n)$. On such a model, standard arithmetic and bitwise boolean operations on word-sized operands can be performed in constant time. Throughout this paper, storage complexities are expressed in terms of bits.

A simple approach to solve the problem is to compute the prefix (or partial) sum array $P[0..n-1]$ such that $P[i]$ stores the sum of $A[0..i]$. Any range sum, $sum(A[i..j])$, can be computed in $O(1)$ time by taking the difference between $P[j]$ and $P[i-1]$. Storing $P[0..n-1]$ requires $O(n \log n)$ bits. This storage requirement can be reduced by way of a succinct (or space-efficient) data structure. The aim of such a structure is to achieve optimal space usage to within lower order additive terms while having asymptotically optimal operation times. In particular, using structures by Raman et al. [12] and Hon et al. [9] with suitable parameters, the partial sum problem can be solved in $O(1)$ time and $kn + o(kn)$ bits.

However, a succinct data structure is still not taking the full opportunity offered by the particular data stored in A. To illustrate this point, suppose there are only $m \ll n$ non-zero entries in A. Then, we can store the indices to those non-zero entries in A together with the associated prefix sums in a predecessor structure. Computing a range sum queries then amounts to two predecessor lookups and an integer subtraction. Using the optimal predecessor structure of Beame and Fich [1], it takes $O(\tau)$ time and $O(m \log n)$ storages where $\tau = \min\left\{\frac{\log\log n}{\log\log\log n}, \sqrt{\frac{\log m}{\log\log m}}\right\}$.

In general, the array A may or may not contain many zeroes. Thus committing ourselves to either a predecessor structure or a succinct data structure without prior knowledge of A may not work well always. Even if we do a scan on A before making the choice, it could happen that A contains few zeroes but is nevertheless rather compressible. To take full advantage of the compressibility of the data, we propose to design data structures whose storage is asymptotically the same as the size of the compressed data while having the same operation times as in a traditional or succinct data structure.

For the range sum query problem here, we design data structures whose space requirement is of the same order as the compressed array when compressed using the Lempel-Ziv algorithm [15] (commonly known as *ZL78*). Our basic structure answers a range sum query in $O(\log n)$ time. With an increase by a constant factor in storage complexity, we obtain a variant in which the query time is $O(\frac{\log\log n}{\log\log\log n} + k)$. Also our method does not modify the compressed data and can be viewed as an additional index structure on it.

Our method is based on the Lempel-Ziv algorithm (obviously) and employed many standard techniques in succinct data structures. The Lempel-Ziv algorithm and its variants are an important class of compression algorithms. Its behaviour is well-understood and is optimal in certain information-theoretic sense. It is popular and easy to implement. For example, it is implemented in the *compress* program in UNIX and in the *arc* program for PC's. In the next section, we will describe the main idea of the Lempel-Ziv algorithm. Then we explain our method for bit strings in sections 3. Section 4 briefly describes the extension to general arrays and some variations. Details will be given in the full paper. Section 5 contains the conclusion.

2 The Lempel-Ziv's Algorithm

The Lempel-Ziv algorithm [15] is a lossless compression algorithm that will automatically adapt to the data distribution. It can be applied to strings over a finite alphabet. For our usage here, we just need to understand the compression and we will omit the decompression. We will first describe the idea on bit strings.

2.1 Parsing the String

The idea is to partition the bit string into *phrases* not appeared before. After marking off the end of the last phrase, we start from the next bit in the input sequence until we come to the shortest string s that has not been marked off before. Denote by s^- the longest prefix of s, i.e., all but the last bit of s. By minimality of $|s|$, s^- has appeared as a phrase before. We mark off s as a new phrase and encode it by the pair (p, b) where p is the index of the phrase s^- (stored in a dictionary of discovered phrases, to be described in section 2.2) and b is the last bit of s. Note that by construction, s^- appears only once before. Moreover, each phrase (of variable length) is now encoded as a fixed length code.

A running example: Consider the following input string. (Spaces are inserted in the string for clarity.)

$$1\ 0\ 01\ 10\ 011\ 11\ 010\ 101\ 00\ 011$$

It will be parsed into the following sequence of phrases:

$$a\ b\ c\ d\ e\ f\ g\ h\ i\ e$$

where the substring represented by each phrase is shown in Figure 1.

phrase	a	b	c	d	e	f	g	h	i
substring represented	1	0	01	10	011	11	010	101	00

Fig. 1. The string represented by each phrase

Note that only the last phrase can be a repetition of some previous phrase. The encoded string is

$$\lambda 1\ \lambda 0\ b1\ a0\ c1\ a1\ c0\ d1\ b0\ e$$

where λ represents the empty string. If there are altogether $c(n)$ distinct phrases, each index p can be encoded in $\log c(n)$ bits. In total, the encoded string has length $c(n)(\log c(n)+1)$ bits (or $(c(n)+1)(\log c(n)+1)-1$ bits if the last phrase has appeared before). It takes $O(c(n)\log c(n))$ bits asymptotically in both cases.

Plugged into the example in Figure 1, the original string has 22 bits. There are 10 phrases in the encoded string, the last one being a repetition of another phrase before. Therefore, the encoded string has $9\times(4+1)+4=49$ bits which is longer

than the original string. For longer strings with many repeating patterns, the phrases will get longer and encoding a long phrase with $\log c(n)$ bits will become a big saving. More rigorously, how good we can compress with Lempel-Ziv is controlled by the parameter $c(n)$. It can be shown that $\sqrt{n} \leq c(n) \leq O(n/\log n)$. Moreover, if we assume the values of A are drawn from a stationary ergodic source with entropy rate H (i.e., it takes on average H bits to describe one value of A in the long run), then $\frac{c(n)\log c(n)}{n} \to H$ as $n \to \infty$. See, for example, Lemma 12.10.1 and Theorem 12.10.1 of Cover and Thomas [4] for more details.

2.2 A Dictionary of Phrases

During the parsing of the input sequence, a dictionary of phrases is gradually built up to facilitate the discovery of new phrases. Newly discovered phrases will be added to the dictionary. An appropriate data structure for the dictionary is the binary trie structure.

A binary trie is a tree in which each internal node has at most two children. The edge to the left (resp. right) child will be labelled with 0 (resp. 1). Each node in the trie corresponds to a string, namely, the string obtained by concatenating all the edge labels in the order from the root to that node. As a special case, the root corresponds to the empty string.

The Lempel-Ziv algorithm will construct a trie T so that each node corresponds to a phrase discovered in the input sequence scanned so far. When we try to discover a new phrase in the remainder of the input sequence, we search T from the root and follow the edges as we read off the bits from the input sequence until we reach a leaf. Then the string formed by appending the next bit in the input sequence to the string represented by that leaf is a new phrase not yet appeared before. We add this to the trie by creating a left or right child to this leaf depending on the next bit is 0 or 1. Thus, parsing requires $O(n)$ time. For our running example, the corresponding trie T is shown in Figure 2.

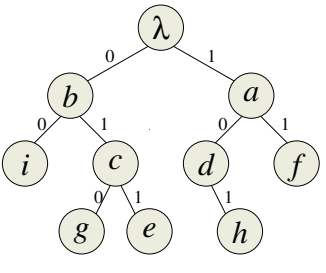


Fig. 2. Trie T

In the ordinary usage of Lempel-Ziv, T is discarded after parsing is completed. Here, we will keep T in order to facilitate the construction of our data structure. After the construction, T is discarded.

3 Range Sum Queries on Compressed Bit Strings

Now we are ready to describe our data structure for bit strings. On the highest level, we partition the input bit string into phrases according to the Lempel-Ziv parsing mentioned in section 2.1. Then we will build two structures:

- the *inter-phrase structure* that supports queries on the number of 1's in a contiguous sequence of phrases, and
- the *intra-phrase structure* that supports queries on the number of 1's in a continuous range within a phrase.

In general, a query range covers a (possibly zero) number of phrases completely and at most two phrases partially. Thus a query range can be broken into at most three parts. See Figure 3. The sequence of completely covered phrases is handled using the inter-phrase structure while the partially covered phrases are handled using the intra-phrase structure.

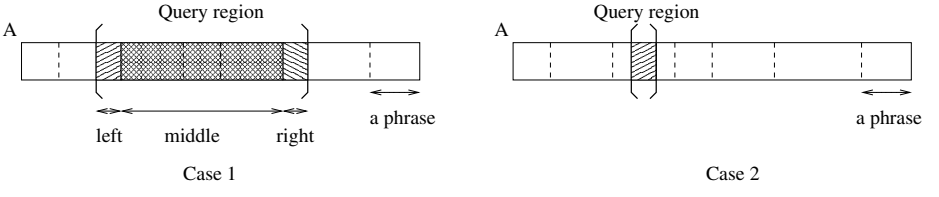


Fig. 3. Breaking down a query region

3.1 The Inter-phrase Structure

The inter-phrase structure contains the arrays P and S. In P, we store in ascending order the starting position of each phrase in the input sequence and in S, we store the number of 1's in front of each phrase. See Figure 4 for an illustration. In total, P and S require $O(c(n) \log n)$ space and can be constructed in $O(n)$ time during the parsing of the input sequence.

P	0	1	2	4	6	9	11	14	17	19
S	0	1	1	2	3	5	7	8	10	10
pointer to phrase	a	b	c	d	e	f	g	h	i	e

Fig. 4. Inter-phrase Structure

Given a query region $[\ell, r]$, we find the smallest i such that $P[i] \geq \ell$ and the largest j such that $P[j] - 1 \leq r$. This takes $O(\log c(n)) = O(\log n)$ time by binary search on P. Then the i-th to $(j-1)$-st phrases are completely within the query range. The number of 1's in this sequence of phrases is computed as $S[j] - S[i]$.

After computing i and j in the above query, we need to query the intra-phrase structure to determine the number of 1's in a range in the $(i-1)$-st and the j-th phrase. To allow for locating any desired phrase in the intra-phrase stucture, the inter-phrase structure will also store an array of pointers (bottom row in Figure 4). It will be clear in the next section that each pointer requires $O(\log c(n))$ bits. Hence the array of pointers requires $O(c(n) \log c(n)) = O(c(n) \log n)$ bits. This array is constructed at the same time when the inter-phrase stucture is being built and it takes $O(c(n))$ time.

3.2 The Intra-phrase Structure

To compute the number of 1's within a range in a phrase quickly, we make use of the trie T. In what follows, we will describe a succinct representation of T in $4c(n) + o(c(n))$ bits, using many standard techniques in succinct data structure design. This will be useful in minimizing the storage in practical implementations. We begin with a few easy definitions.

Definition 1: The *depth* of a node in the trie is defined as the number of edges from the root to that node.

Definition 2: The *level-d ancestor* of a node v is the (unique) node that has depth d on the path from the root to v.

Definition 3: The *Euler tour* of a rooted tree T is the sequence of edges traversed during a depth-first traversal of T starting from the root, and ending at the root after visiting all the nodes in T.

For example, the Euler tour of our trie T is shown on the row labelled with E in Figure 5. Here, we represent an edge by one of the 4 symbols, $(_0$, $(_1$, $)_0$ and $)_1$, depending on whether the edge is downward or upward and with label 0 or 1. Since each edge is traversed twice, this sequence contains $2c(n)$ brackets in total. Furthermore, each bracket is associated with a node, namely, the node reached by following that edge. See the row labelled with "phrase" in Figure 5. Notice that a node may appear more than once in the Euler tour.

phrase	b	i	b	c	g	c	e	c	b	λ	a	d	b	d	a	f	a	λ
E	$(_0$	$(_0$	$)_0$	$(_1$	$(_0$	$)_0$	$(_1$	$)_1$	$)_1$	$)_0$	$(_1$	$(_0$	$(_1$	$)_1$	$)_0$	$(_1$	$)_1$	$)_1$
D	$+1$	$+1$	-1	$+1$	$+1$	-1	$+1$	-1	-1	-1	$+1$	$+1$	$+1$	-1	-1	$+1$	-1	-1
C	-1	-1	$+1$	$+1$	-1	$+1$	$+1$	-1	-1	$+1$	$+1$	-1	$+1$	-1	$+1$	$+1$	-1	-1

Fig. 5. Intra-phrase Structure

Definition 4: An *index* of a node v in a tree T is the position of an occurrence of v in the Euler tour of T, and the *principle index* is the position of the leftmost occurrence.

The intra-phrase structure will support the following primitive operations:

Count(v) – Given an index of a node v, find the number of 1's on the path from the root to v.
Depth(v) – Given an index of a node v, find the depth of v, i.e., the number of edges from the root to v.
Ancestor(v, d) – Given the principle index of a node v and an integer d, find an index of the level-d ancestor of v.

Given these three operations, we can compute the number of 1's on the path from the level-i ancestor to the level-j ancestor of a node v, given i, j and the principle index of node v. That is, we compute $Count(Ancestor(v, j)) - Count(Ancestor(v, i-1))$. This corresponds to the number of 1's in the continuous range from position i to j of a phrase v. To be able to invoke *Ancestor*(), the inter-phrase structure stores the principle indices of the phrases in the array of pointers (bottom row of Figure 4). Each index requires $O(\log c(n))$ bits.

To support the three primitive operations, we store two arrays, C and D, of bits. See Figure 5. (Note that we do not store E and the row labelled with "phrase".) The array D will give information on the depth of the nodes in the trie while C and D together will give information on the number of 1's along the path from the root to each node.

More precisely, for $0 \le i < 2c(n)$, we define $D[i]$ as $+1$ (-1) if $E[i]$ is an open (close) bracket. Hence, the sum of values in $D[0..i]$ is the difference between the number of open and close brackets in $E[0..i]$. This, in turn, represents the depth of the node reached by following the traversal specified in $E[0..i]$. Similarly, for $0 \le i < 2c(n)$, we define $C[i]$ as

$$C[i] = \begin{cases} +1 \text{ if } & E[i] =)_0 \text{ or } (_1 \\ -1 \text{ if } & E[i] =)_1 \text{ or } (_0 \end{cases}.$$

It is easy to check that

$$C[i] + D[i] = \begin{cases} +2 \text{ if } E[i] = (_1 \\ -2 \text{ if } E[i] =)_1 \\ 0 \quad \text{if } E[i] =)_0 \text{ or } (_0 \end{cases}.$$

Hence the sum of values in $C[0..i]$ and $D[0..i]$ gives twice the number of 1's from the root to the node corresponding to $E[i]$.

Supporting *Count*(v) and *Depth*(v). We need to compute the sum of values in $C[0..i]$ and $D[0..i]$ where i is an index of v in E. We will explain the computation for $C[0..i]$. (Computaion for $D[0..i]$ is identical.) The technique is typical in succinct data structures.

We will construct an array C^0 with $\frac{2c(n)}{\log^2 c(n)} = o(c(n))$ entries such that for $i = 0$ to $\frac{2c(n)}{\log^2 c(n)} - 1$, the entry $C^0[i]$ will store the sum in $C\left[0..(i+1)\log^2 c(n) - 1\right]$. Since C has length $2c(n)$, the maximum sum can be stored in $O(\log c(n))$ bits. Hence C^0 will occupy $O\left(\frac{c(n)}{\log^2 c(n)} \times \log c(n)\right) = o(c(n))$ bits. We can, in $O(1)$ time, look up the number of 1's in $C[0..i]$ when i is a multiple of $\log^2 c(n)$.

Next, we will construct another array C^1 with $\frac{4c(n)}{\log c(n)}$ entries. For $i = 0$ to $\frac{2c(n)}{\log^2 c(n)} - 1$ and for $j = 0$ to $2\log c(n) - 1$, the entry $C^1[i \cdot 2\log c(n) + j]$ will contain the sum in $C\left[i\log^2 c(n) \ldots i\log^2 c(n) + (j+1)\frac{\log c(n)}{2} - 1\right]$. Since each short-ranged sum is at most $\log^2 c(n)$, each entry of C^1 requires only $\log(\log^2 c(n))$ bits. Hence C^1 occupies $O\left(\frac{4c(n)}{\log c(n)} \times \log\log c(n)\right) = o(c(n))$ bits. With C^1 and C^0, we can compute the sum in $C[0..i]$ when i is a multiple of $\frac{\log c(n)}{2}$.

Finally, for the number of 1's within a (± 1)-pattern of length $\frac{\log c(n)}{2}$, we make use of a 2-dimensional lookup table L. Notice that there are $2^{(\log c(n))/2} = \sqrt{c(n)}$ different $(\pm)$-patterns of length $\frac{\log c(n)}{2}$. For each such pattern α and for each position $0 \le j < \frac{\log c(n)}{2}$, the sum of ± 1's from position 0 to j can be precomputed and stored in $L[\alpha, j]$. Thus, L has size $\sqrt{c(n)} \times \frac{\log c(n)}{2} \times \log\log c(n) = o(c(n))$ bits. Moreover, each chunk of $\log c(n)/2$ bits in C can share the same table L.

We will similarly construct D^0 and D^1 for D but we can share the same lookup table L. In total, the storage required by the arrays C, C^0, C^1, D, D^0, D^1 and L is $4c(n) + o(c(n))$ bits. Moreover, the operations require only constant time to complete.

Supporting *Ancestor*(v, d). Observe that the lowest common ancestor (LCA) of v with any node is an ancestor of v. Let j be the principle index of v. By the property of an Euler tour, the LCA of the nodes with indices x and y $(x \le y)$ is the node of minimum depth among those with an index z such that $x \le z \le y$. Thus, for any $i \le j$, the LCA of node v and the node corresponding to i is the node with minimum depth in $E[i..j]$. It can be proved that the depth of the LCA between the nodes with indices i and j is monotonic increasing as i increases from 0 to j. Details will be given in the full paper. Thus the level-d ancestor of j can be found by a binary search in $O(\log c(n)) = O(\log n)$ time, provided we can compute the depth of the LCA between any pair of indices, (i, j), in constant time. This amounts to computing the minimum among $D[0..i], D[0..i+1], \ldots, D[0..j]$. In the next subsection, we will describe a data structure of space $o(c(n))$ that answers such implicit range minimum queries in $O(1)$ time. There exist algorithms that support level ancestor query in $O(1)$ time using space $O(n \log n)$ [2] or even $O(n)$ [6]. We do not use them here because the overall query time is not improved and they do not blend well with the other components of our structures, resulting in a larger constant factor in the storage.

3.3 A Range Minimum Structure

Let $\tilde{D}_1$ be the array storing the index to the minimum of each sub-interval of length $\log^3 c(n)$ in D. Thus $\tilde{D}_1$ has $2c(n)/\log^3 c(n)$ entries, each storing an $O(\log c(n))$-bit index. On this array, we build an APM structure of Poon [11] which can answer range min queries in constant time and uses $\frac{2c(n)}{\log^3 c(n)} \log\left(\frac{2c(n)}{\log^3 c(n)}\right) \log c(n) = o(c(n))$ bits. We denote this structure by $APM(\tilde{D}_1)$.

For each interval of length $\log^3 c(n)$ in D, we further break it down into sub-intervals of length $\log c(n)/2$ and store the index to the minimum of each

sub-interval in another array $\tilde{D}_2$ of length $2c(n)/\log^3 c(n) \times 2\log^2 c(n) = 4c(n)/\log c(n)$. This time, each index will need only $O(\log\log c(n))$ bits since they are indices relative to a sub-interval. Again, we build an APM structure for every $2\log^2 c(n)$ entries of $\tilde{D}_2$. We denote this collection of structures by $APM(\tilde{D}_2)$. This requires $\frac{2c(n)}{\log^3 c(n)}(2\log^2 c(n)\log(2\log^2 c(n))\log\log c) = o(c(n))$ bits.

Finally, we build a lookup table M for range minimum within a sub-interval of length $\log c(n)/2$. The table has size $(\sqrt{c(n)}) \times (\log c(n)/2)^2 \times \log\log c(n)$ bits. The table is shared by all the sub-intervals of length $(\log c(n))/2$. In total, the two structures, $APM(\tilde{D}_1)$ and $APM(\tilde{D}_2)$, together with M, require at most $o(c(n))$ bits.

4 Extension and Variations

4.1 Generalizing to Array of Integers

To extend the previous ideas on a general array of k-bit integers, we apply Lempel-Ziv over an alphabet of size 2^k. Thus, the dictionary of phrases T will be a 2^k-ary trie instead of a binary trie.

For the inter-phrase structure, we have P, S and the array of principle indices using $O(c(n)\log n)$ bits of storage. For the intra-phrase structure, we store T succinctly using array D as defined before but with arrays $C_0, C_1, \ldots, C_{k-1}$ defined as follows. For each $i \in \{0, \ldots, k-1\}$, C_i is designed such that the sum of values in $C_i[0..j]$ and $D[0..j]$ is equal to twice the number of 1's in the i-th bit position of the phrase from the root to the node corresponding to $E[i]$. Furthermore, we will store $APM(\tilde{D}_1)$ and $APM(\tilde{D}_2)$. In total, these require $(2(k+1)c(n) + o(c(n)) = O(c(n)\log n)$ bits of storage as $k = O(\log n)$.

To handle a query, we first query the inter-phrase structure as in section 3.1. For the intra-phrase query, we compute the number of 1's within the required range in each of the k bit positions. Then we shift these k sums properly and sum them. This takes $O(k) = O(\log n)$ time.

4.2 Speeding Up Queries

To speed up the query, we use an optimal predecessor structure [1] to store the P array. Furthermore, we will have a number of predecessor structures, one for each level of T, to store entries in the Euler tour E. It can be proved that the level-d ancestor of a node v is the predecessor (among those indices of nodes in level d of T) of i where i is the principle index of v. Thus, both the inter-phrase and intra-phrase queries can be sped up to $O(\frac{\log\log n}{\log\log\log n})$ time while the storage complexity remains to be $O(c(n)\log n)$. Hence for arrays of k-bit integers, the query time is $O(\frac{\log\log n}{\log\log\log n} + k)$.

4.3 Reducing Storage

Although in terms of asymptotic complexity, the storage is optimal relative to the Lempel-Ziv compression, the constant factor can still be reduced. This will be

important in practice. The idea is to sparsify the arrays P and S by storing only part of the entries. This will reduce the storage from $3c(n)\log n + 4c(n) + o(c(n))$ to $c(n)\log c(n) + 6c(n) + o(c(n))$.

5 Conclusion

We have described an opportunistic data structure that support efficient range sum queries in a compressed sequence of integers. In fact, many variations in implementations are possible. We have implemented the version without the predecessor structures since the increase in the constant factor in storage due to the predecessor structures makes it less attractive in practice.

References

1. P. Beame and F. Fich. Optimal bounds for the predecessor problem. In *STOC'99*, pages 295–304, 1999.
2. M. Bender and M. Farach-Colton. The level ancestor problem simplified. In *LATIN'02*, *LNCS 2286*, pages 508–512, April 2002.
3. M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithms. Technical Report 124, Digital SRC Research Report, 1994.
4. Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications. John Wiley & Sons, 1991.
5. Paolo Ferragina and Giovanni Manzini. On compressing and indexing data. Technical Report TR-02-01, Dipartimento di Informatica, Universita di Pisa, Jan 2002.
6. Richard Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. In *SODA'04*, pages 1–10, 2004.
7. S.W. Golomb. Run-length encodings. *IEEE Transaction on Information Theory*, 12:399–401, 1966.
8. Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *STOC'00*, pages 397–406, 2000.
9. Wing-Kai Hon, Kunihiko Sadakane, and Wing-Kin Sung. Succinct data structures for searchable partial sums. In *ISAAC'03*, *LNCS 2906*, pages 505–516, 2003.
10. D.A. Huffman. A method for the construction of minimum-redundancy codes. In *Proc. of the I.R.E. 40*, pages 1098–1101, 1952.
11. Chung Keung Poon. Dynamic orthogonal range queries in OLAP. *Theoretical Computer Science*, 296(3):487–510, March 2003.
12. Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct dynamic data structures. In *WADS'01*, *LNCS 2125*, pages 426–437, 2001.
13. K. Sadakane. Succinct representation of lcp information and improvements in the compressed suffix arrays. In *SODA'02*, pages 225–232, 2002.
14. K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *Journal of Algorithms*, 48:294–313, 2003.
15. J. Ziv and A. Lempel. Compression of individual sequences by variable rate coding. *IEEE Transaction on Information Theory*, 24(5):530–536, 1978.

Generating Combinations by Prefix Shifts

Frank Ruskey* and Aaron Williams**

Dept. of Computer Science, University of Victoria

Abstract. We present a new Gray code for combinations that is practical and elegant. We represent combinations as bitstrings with s 0's and t 1's, and generate them with a remarkably simple rule: Identify the shortest prefix ending in 010 or 011 (or the entire bitstring if no such prefix exists) and then rotate (shift) it by one position to the right. Since the rotated portion of the string consists of at most four contiguous runs of 0's and 1's, each successive combination can be generated by transposing only one or two pairs of bits. This leads to a very efficient loopless implementation. The Gray code also has a simple and efficient ranking algorithm that closely resembles that of combinations in colex order. For this reason, we have given a nickname to our order: cool-lex!

1 Background and Motivation

An important class of computational tasks is the listing of fundamental combinatorial structures such as permutations, combinations, trees, and so on. Regarding combinations, Donald E. Knuth [8] writes "Even the apparently lowly topic of combination generation turns out to be surprisingly rich, ... I strongly believe in building up a firm foundation, so I have discussed this topic much more thoroughly than I will be able to do with material that is newer or less basic."

The applications of combination generation are numerous and varied, and Gray codes for them are particularly valuable. We mention as application areas cryptography (where they have been implemented in hardware at NSA), genetic algorithms, software and hardware testing, statistical computation (e.g., for the bootstrap, Diaconis and Holmes [3]), and, of course, exhaustive combinatorial searches.

As is common, combinations are represented as bitstrings of length $n = s + t$ containing s 0's and t 1's. We denote this set as $\mathbf{B}(s,t) = \{b_1 b_2 \cdots b_n \mid \sum b_i = t\}$. Another way of representing combinations is as increasing sequences of the elements in the combination. We denote this set as $\mathbf{C}(s,t) = \{c_1 c_2 \cdots c_t \mid 1 \leq c_1 < c_2 < \cdots < c_t \leq s + t\}$.

We consider here the problem of listing the elements of $\mathbf{B}(s,t)$ so that successive bitstrings differ by a prefix that is cyclically shifted by one position to the right. We call such shifts prefix shifts, or rotations, and they may be represented by a cyclic permutation $\sigma_k = (1\ 2\ \cdots\ k)$ for some $2 \leq k \leq n$, where this permutation acts on the indices of a bitstring.

* Research supported in part by an NSERC Discovery Grant.
** Research supported in part by a NSERC PGS-D.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 570–576, 2005.

As far as we are aware, the only other class of strings that has a listing by prefix shifts are permutations, say of $\{1, 2, \ldots, n\}$. In Corbett [1] and Jiang and Ruskey [7] it is shown that all permutations may be listed by prefix shifts. That is, the directed Cayley graph with generators $(1\ 2), (1\ 2\ 3), \ldots, (1\ 2\ \cdots\ n)$ is Hamiltonian. In our case we have the same set of generators acting on the indices of the bitstring, but the underlying graph is not vertex-transitive; in fact, it is not regular.

There are many algorithms for generating combinations. The one presented here has the following novel characteristics.

1. Successive combinations differ by a prefix shift. There is no other algorithm for generating combinations with this feature. In some applications combinations are represented in a single computer word; our algorithm is very fast in this scenario. It is also very suitable for hardware implementation.
2. Successive combinations differ by one or two transpositions of a 0 and a 1. There are other algorithms where successive combinations differ by a single transposition (Tang and Liu [10]). Furthermore, that transposition can be further restricted in various ways. For example, so that only 0's are between the transposed bits (Eades and McKay) [5], or such that that the transposed bits are adjacent or have only one bit between (Chase [2]). Along with ours, these other variants are ably discussed in Knuth [8].
3. The list is circular; the first and last bitstrings differ by a prefix shift.
4. The list for (s, t) begins with the list for $(s-1, t)$. Usually, this property is incompatible with Property 3, relative to the elementary operation used to transform one string to the next. For example, colex order has Property 4 but not Property 3.
5. The algorithm can be implemented so that *in the worst case* only a small number of operations are done between successive combinations, *independent* of s and t. Such algorithms are said to be loopless, an expression coined by Ehrlich [6].
6. Unlike other Gray codes for combinations, this one has a simple ranking function whose running time is $O(n)$ arithmetic operations.

2 Recursive Construction Rule

If $S = s_1, s_2, \ldots, s_m$ is a sequence of strings and x is a symbol, then Sx represents the sequence of strings $Sx = s_1x, s_2x, \ldots, s_mx$. We recursively define the following list of bitstrings.

$$\mathbf{W}_{st} = \mathbf{W}_{(s-1)t}0,\ \mathbf{W}_{s(t-1)}1,\ 1^{t-1}0^s1 \tag{1}$$

As will be proven below this list accounts for all strings in $\mathbf{B}(s, t)$ except for 1^t0^s. To get all of $\mathbf{B}(s, t)$ we define

$$\mathbf{W}'_{st} = 1^t0^s, \mathbf{W}_{st}. \tag{2}$$

Examples of $\mathbf{W}'$ may be found in Figure 1 (the additional columns for $\mathbf{W}'_{33}$ give the corresponding element of $\mathbf{C}(3, 3)$ and the rotation σ_k used in transforming one bitstring to the next).

$\mathbf{W}'_{42}$	$\mathbf{W}'_{24}$	$\mathbf{W}'_{33}$		
110000	111100	111000	123	σ_4
011000	011110	011100	234	σ_2
101000	101110	101100	134	σ_3
010100	110110	110100	124	σ_5
001100	111010	011010	235	σ_4
100100	011101	101010	135	σ_4
010010	101101	010110	245	σ_3
001010	110101	001110	345	σ_3
000110	011011	100110	145	σ_4
100010	101011	110010	125	σ_6
010001	010111	011001	236	σ_2
001001	001111	101001	136	σ_4
000101	100111	010101	246	σ_3
000011	110011	001101	346	σ_3
100001	111001	100101	146	σ_5
		010011	256	σ_3
		001011	356	σ_4
		000111	456	σ_4
		100011	156	σ_5
		110001	126	σ_6

Fig. 1. Cool-lex listings $\mathbf{W}'_{42}$, $\mathbf{W}'_{24}$, $\mathbf{W}'_{33}$.

Theorem 1. *The list* $\mathbf{W}_{st}$ *defined in (1) has the following properties.*

- *The list contains each bitstring of* $\mathbf{B}(s,t)$ *exactly once, except for* 1^t0^s.
- *Successive bitstrings differ by a prefix shift of one position to the right.*
- *Successive bitstrings differ by the transposition of one or two pairs of bits.*
- $first(\mathbf{W}_{st}) = 01^t0^{s-1}$.
- $last(\mathbf{W}_{st}) = 1^{t-1}0^s1$.

Proof. Our proof is by induction on $n = s + t$. The first property is satisfied since, inductively, $\mathbf{W}_{(s-1)t}0$ is a list of all elements of $\mathbf{B}(s,t)$ that end in a 0, except for $1^t0^{s-1}0$, and $\mathbf{W}_{s(t-1)}1$ is a list of all elements of $\mathbf{B}(s,t)$ that end in a 1, except for the bitstring $1^{t-1}0^t1$, which is appended to the end of the list.

To prove the remaining properties it is convenient to separate out the cases $t = 1$ and $s = 1$. The interfaces between sublists are indicated below as horizontal lines, and transposed bits are underlined. The list below on the left is for $t = 1$ and on the right for $s = 1$.

$$\begin{array}{l} 010^{s-2}\ 0 \\ \vdots \\ \underline{0^{s-2}01}\ \underline{0} \\ \hline\hline 00^{s-2}0\ 1 \end{array} \qquad \begin{array}{l} \overline{011^{t-2}\ 1} \\ \vdots \\ \underline{1^{t-2}01}\ 1 \\ \hline 11^{t-2}0\ 1 \end{array}$$

Below we show the lists for the case where $t > 1$ and $s > 1$. The left and right lists are identical, except that the left list illustrates shifts, while the right

list illustrates transpositions. The starting and ending bitstring in each sublist is obtained from the induction assumption.

$$\begin{array}{ll} 011^{t-2}10^{s-2}\ 0 & 011^{t-2}10^{s-2}\ 0 \\ \vdots & \vdots \\ \underline{11^{t-2}00^{s-2}1\ 0} & \underline{1}1^{t-2}\underline{0}0^{s-2}\underline{1}\ \underline{0} \\ 011^{t-2}00^{s-2}\ 1 & 01^{t-2}10^{s-2}0\ 1 \\ \vdots & \vdots \\ \underline{1^{t-2}00^{s-2}01\ 1} & \underline{1^{t-2}0}00^{s-2}\underline{1}\ 1 \\ 11^{t-2}00^{s-2}0\ 1 & 1^{t-2}100^{s-2}0\ 1 \end{array}$$

To verify the second and third properties we need to examine what happens at the interfaces (indicated by the long horizontal lines) between the lists in (1) as illustrated above. Note that at the two interfaces the successive bitstrings differ by a right rotation of all n positions (although at the second interface we could also think of it as a rotation of the first $n-1$ positions).

Note that two pairs of bits are transposed at the first interface, and one pair of bits at the second interface. Thus the third property is satisfied. Finally, observe that the first and last bitstrings in these lists have the required form. □

To close this section, we observe that $last(W_{st}) = \overline{first(W_{ts})}^{R}$; such equations would allow us to define cool-lex order in other ways.

3 Implementation

Referring back to the proof of Theorem 1, we observe that the bits that are transposed at the first interface are at positions $(1, t)$ and $(n-1, n)$, and at the second interface are at positions $(t-1, n-1)$. Below we show a recursive implementation of the algorithm; this is followed by an iterative implementation. In both cases, the code that initializes `b` to 1^t0^s and outputs it is omitted; we also assume that $s > 0$ and $t > 0$.

For the recursive version, the array `b` has indexing starting at 1. The initial call is `swap( 1, t+1 ); visit( b ); gen( s, t );`. Since every recursive call is followed by a visit, the algorithm runs in constant amortized time.

```
static void gen ( int s, int t ) {
     if (s > 1) { gen( s-1, t );
         swap( 1, t );  swap( s+t, s+t-1 ); visit( b ); }
     if (t > 1) { gen( s, t-1 );
         swap( t-1, s+t-1 );  visit( b ); }
}
```

We now present the iterative loopless implementation. In this case the array indexing is 0 based. It is useful to maintain a variable x, which is the smallest index for which `b[x-1] == 0` and `b[x] == 1`. In terms of shifts, the code to obtain the next bitstring and to update `x` is amazingly simple.

```
shift( ++x );
if (b[0] == 0 && b[1] == 1) x = 1;
```

To generate the next bitstring by transpositions it is useful to maintain another variable y, which is the smallest index for which `b[y] == 0`. Referring back to the proof of Theorem 1 we observe that in every case `b[x]` becomes `0` and `b[y]` becomes `1`. The test `b[x+1] == 0` determines whether we are at the first or the second interface. If we are at the first interface, then we set `b[x+1]` to `1` and `b[0]` to `0`. It now remains to update x and y. At the second interface they are simply incremented. At the first interface y always becomes 0; also, x is incremented unless $y = 0$, in which case x becomes 1 (see the $t = 1$ case of the proof of Theorem 1).

```
static void iterate ( int s, int t ) {
   b[t] = 1;  b[0] = 0;
   visit( b );
   int x = 1, y = 0;
   while (x < n-1) {
      b[x++] = 0;  b[y++] = 1;      /* X(s,t) */
      if (b[x] == 0) {
         b[x] = 1;  b[0] = 0;       /* Y(s,t) */
         if (y > 1) x = 1;          /* Z(s,t) */
         y = 0; }
      visit( b ); } }
```

The structure of the implementation allows us to completely determine the number of times each statement in the code is executed. Call the relevant quantities $X(s,t)$, $Y(s,t)$, and $Z(s,t)$ corresponding to the various statements as shown above. I.e., $Y(s,t)$ is the number of times `b[x] == 0` is true and $Z(s,t)$ is the number of times `y > 1` is true. We find that

$$X(s,t) = \binom{s+t}{t} - 1, \quad Y(s,t) = \binom{s+t-1}{t}, \quad Z(s,t) = \binom{s+t-2}{t-1}.$$

4 Ranking Algorithm

Given a listing of combinatorial structures, the *rank* of a particular structure is the number of structures that precede it in the listing.

Colex order is lexicographic order applied to the reversal of strings. It has many uses, for example in Frankl's now standard proof of the Kruskal-Katona Theorem [11]. Given an (s,t)-combination represented as a bitstring $b_1 b_2 \cdots b_n$ the corresponding set elements can be listed as $c_1 < c_2 < \cdots < c_t$ where c_i is the position of the i-th 1 in the bitstring. As is well-known ([8],[11]) in colex order the rank of $c_1 c_2 \cdots c_t$ is

$$\sum_{j=1}^{t} \binom{c_i - 1}{i}. \tag{3}$$

As we see in the statement of the theorem below, in cool-lex order there is a very similar rank function. Let $rank(c_1c_2\cdots c_t)$ denote the rank of $c_1c_2\cdots c_t \in \mathbf{C}(s,t)$ in our order.

Theorem 2. *Let r be the smallest index such that $c_r > r$ (so that $c_{r-1} = r-1$).*

$$rank(c_1c_2\cdots c_t) = \binom{c_r}{r} - 1 + \sum_{j=r+1}^{t} \left(\binom{c_j - 1}{j} - 1 \right), \tag{4}$$

Proof. Directly from the recursive construction (1) we have

$$rank(b_1b_2\cdots b_n) = \begin{cases} rank(b_1b_2\cdots b_{n-1}) & \text{if } b_n = 0, \\ \binom{s+t}{t} - 1 & \text{if } b_1b_2\cdots b_n = 1^{t-1}0^s1, \\ \binom{s+t-1}{t-1} - 1 + rank(b_1b_2\cdots b_{n-1}) & \text{otherwise.} \end{cases}$$

Let us now consider the rank in terms of the corresponding list of elements $1 \le c_1 < c_2 < \cdots < c_t$. The case $rank(b_1b_2\cdots b_n) = rank(b_1b_2\cdots b_{n-1}) = rank(b_1b_2\cdots b_{n-2})$ will continue to apply until $b_{n-k} = 1$; i.e., until $n-k = c_{t-1}$. Hence the number of 0's and 1's to the left of position c_{t-1} in $b_1b_2\cdots b_n$ is $c_{t-1} - 1$, which leads us to the expression below.

$$rank(c_1c_2\cdots c_t) = \begin{cases} \binom{c_t}{t} - 1 & \text{if } c_t = n \text{ and } c_{t-1} = t-1 \\ \binom{c_{t-1}-1}{t-1} - 1 + rank(c_1c_2\cdots c_{t-1}) & \text{otherwise.} \end{cases}$$

As in (3) and (4), the recursion above has the remarkable and useful property that it depends only on t and not on s. In other words, the cool-lex lists begin with cool-lex lists with smaller s values (fewer zeroes). □

The ranking function can also be written recursively, as shown below.

$$rank(c_1c_t\cdots c_t) = rank(c_1c_2\cdots c_{t-1}) + \binom{c_r - 1}{r - 1} + r - t - 1. \tag{5}$$

Using standard techniques, as explained for example in [8] the expression in (4) can be evaluated in $O(n)$ arithmetic operations.

5 Final Remarks

Unlike every other recursive Gray code definition, (1) has the remarkable property that it involves no reversal of lists. The list for $C(6,3)$ has been rendered musically by George Tzanetakis and is available on the page `http://www.cs.uvic.ca/~ruskey/Publications/Coollex/Coollex.html`

The algorithm discussed here appears in Knuth's prefasicle [8] (latest version of January 19, 2005). The output of the algorithm is illustrated in Figure 26 on page 16. He refers to the listing as suffix-rotated (since he indexes the bitstrings $b_{n-1}\cdots b_1b_0$). See also Exercise 55 on page 30 and it's solution on page 46.

To conclude the paper we list some open problems:

- Is it possible to generate combinations if the allowed operations are futher restricted? For example, all permutations can be generated by letting the permutations $(1\ 2)$ and $(1\ 2\ \cdots\ n)$ and their inverses act on the indices. Can all combinations be so generated?
- Can the permutations of a multiset be generated by suffix rotations?
- What is the fastest combination generator when carefully implemented? It would be interesting to undertake a comparative evaluation in a controlled environment, say of carefully implemented MMIX programs. Testing should be done, in the three cases, depending on whether the combination is represented by a single computer word, an element of $\mathbf{B}(s,t)$, or an element of $\mathbf{C}(s,t)$.

References

1. P.F. Corbett, *Rotator Graphs: An Efficient Topology for Point-to-Point Multiprocessor Networks*, IEEE Transactions on Parallel and Distributed Systems, 3 (1992) 622–626
2. P.J. Chase, *Combination Generation and Graylex Ordering*, Congressus Numerantium, 69 (1989) 215–242.
3. P. Diaconis and S. Holmes, *Gray codes for randomization procedures*, Statistical Computing, 4 (1994) 207–302.
4. P. Eades, M. Hickey and R. Read, *Some Hamilton Paths and a Minimal Change Algorithm*, Journal of the ACM, 31 (1984) 19–29.
5. P. Eades and B. McKay, *An Algorithm for Generating Subsets of Fixed Size with a Strong Minimal Change Property*, Information Processing Letters, 19 (1984) 131–133.
6. G. Ehrlich, *Loopless Algorithms for Generating Permutations, Combinations and Other Combinatorial Configurations*, Journal of the ACM, 20 (1973) 500–513.
7. M. Jiang and F. Ruskey, *Determining the Hamilton-connectedness of certain vertex-transitive graphs*, Discrete Mathematics, 133 (1994) 159–170.
8. Donald E. Knuth, *The Art of Computer Programming*, pre-fascicle 4A (a draft of Section 7.2.1.3: Generating all Combinations), Addison-Wesley, 2004, 61 pages (`http://www-cs-faculty.stanford.edu/~knuth/fasc3a.ps.gz`).
9. F. Ruskey, *Simple combinatorial Gray codes constructed by reversing sublists*, 4th ISAAC (International Symposium on Algorithms and Computation), Lecture Notes in Computer Science, #762 (1993) 201–208.
10. D.T. Tang and C.N. Liu *Distance-2 Cycle Chaining of Constant Weight Codes*, IEEE Transactions, **C-22** (1973) 176–180.
11. D. Stanton and D. White, *Constructive Combinatorics*, Springer-Verlag, 1986.

Error-Set Codes and Related Objects*

An Braeken[1], Ventzislav Nikov[2], and Svetla Nikova[1]

[1] Department Electrical Engineering, ESAT/COSIC
Katholieke Universiteit Leuven, Kasteelpark Arenberg 10
B-3001 Heverlee-Leuven, Belgium
{an.braeken,svetla.nikova}@kuleuven.ac.be
[2] Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
v.nikov@tue.nl

Abstract. By considering a new metric, Nikov and Nikova defined the class of error-set correcting codes. These codes differ from the error-correcting codes in the sense that the minimum distance of the code is replaced by a collection of monotone decreasing sets Δ which define the supports of the vectors that do not belong to the code. In this paper we consider a subclass of these codes - so called, ideal codes - investigating their properties such as the relation with its dual and a formula for the weight enumerator. Next we show that the Δ-set of these codes corresponds to the independent sets of a matroid. Consequently, this completes the equivalence of ideal linear secret sharing schemes and matroids on one hand and linear secret sharing schemes and error-set correcting codes on the other hand.

1 Introduction

Nikov and Nikova introduced a class of generalized codes, called error-set codes in [14]. These codes were originally defined by the property that the codewords should belong to a monotone increasing set. In this paper, we show that the ideal error-set codes can be represented as $[N, k, \Delta]$-code, where N is the length, k the corresponding dimension and where the monotone decreasing set Δ defines the forbidden supports of the codewords (forbidden distances). Error-set codes have been constructed by means of Monotone Span Programs (MSP) and have been used in order to establish the minimum conditions for security of linear secret sharing schemes (LSSS) and verifiable secret sharing (VSS) schemes.

This paper shows that the set of forbidden distances Δ of the ideal error-set codes corresponds to the independent sets of a matroid. From this relation, we derive other properties and insights of the error-set codes. For instance, we show

* The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT and by Concerted Research Action GOA Ambiorix 2005/11 of the Flemish Government. An Braeken is research assistent of the FWO.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 577–585, 2005.

how the corresponding dual error-set code is constructed from a given error-set code. The equivalence between ideal LSSS and matroids was known since 1991 [1]. In 2003 [14], the equivalence between LSSS and the error-set codes has been proven. Consequently, the relation between ideal error-set codes and matroids follows.

The paper is organized as follows. In Sect. 2 we start with the definitions of matroids and the relation between linear codes and matroids. In Sect. 3, we study the properties of ideal error-set codes. In Sect. 4, we recall the relations between ideal LSSS and matroids, LSSS and codes, and show how the relation between matroids and ideal codes completes the equivalence relations. We end with some conclusions in Sect. 5.

2 Background

We first explain the definitions and properties of matroids together with a relation between matroids and error-correcting codes.

2.1 Matroids

Matroids have been introduced by Withney [16]. We first recall some basic facts, but refer to [17] for a more comprehensive introduction into the subject. A *matroid* $\mathcal{M} = (\mathcal{S}, \mathcal{I})$ is a finite set $\mathcal{S}$ and a collection $\mathcal{I}$ of subsets of S (called the *independent* sets) such that $(I1) - (I3)$ are satisfied:

$(I1)$ $\emptyset \in \mathcal{I}$.
$(I2)$ If $X \in \mathcal{I}$ and $Y \subseteq X$, then $Y \in \mathcal{I}$.
$(I3)$ If $U, V \in \mathcal{I}$ with $|U| = |V| + 1$, there exists $x \in U \setminus V$ such that $V \cup x \in \mathcal{I}$.

A subset of $\mathcal{S}$ not belonging to $\mathcal{I}$ is called *dependent*. The maximal independent subsets of $\mathcal{S}$ are called the *bases*, while the minimal dependent subsets are called *circuits*. The collection of bases is denoted by $\mathcal{B}$, while the collection of circuits is denoted by $\mathcal{C}$. A matroid is uniquely defined by $\mathcal{B}$ or $\mathcal{C}$.

The following theorem, known as the augmentation theorem, gives as a consequence that all bases in $\mathcal{M}$ have the same cardinality.

Theorem 1. *[17] (Augmentation Theorem) Suppose that $X, Y \in \mathcal{I}$ and that $|X| < |Y|$. Then there exists $Z \subseteq Y \setminus X$ such that $|X \cup Z| = |Y|$ and $X \cup Z \in \mathcal{I}$.*

If every set of cardinality k is a base in $\mathcal{S}$ with $1 \leq k \leq n$ where n is the number of elements of $\mathcal{S}$, the matroid is said to be *uniform*, denoted by $\mathcal{M}_{k,n}$.

A matroid can also be uniquely defined by its *rank function*

$$\rho : 2^{\mathcal{S}} \rightarrow \mathbb{Z} : \quad \rho(A) = \max\{|X| : X \subseteq A, X \in \mathcal{I}\}, \quad \forall A \subseteq \mathcal{S}.$$

The *rank of the matroid*, denoted by $r(\mathcal{M})$ is the rank of the bases. Consequently, if C is a circuit then $\rho(C) = |C| - 1$ and every proper subset of a circuit is independent. If for $x \in \mathcal{S}$, $A \subseteq \mathcal{S}$, $\rho(A \cup x) = \rho(A)$ it is said that x *depends* on A and it is denoted by $x \sim A$. With any matroid, one can associate its dual.

Definition 1. *[17] If $\mathcal{B} = \{B_i : i \in I\}$ is the set of bases of a matroid $\mathcal{M}$. Then the dual matroid $\mathcal{M}^*$ of a matroid $\mathcal{M}$ is defined by the set of bases $\mathcal{B}^* = \{\mathcal{S} \setminus B_i : i \in I\}$. The corresponding rank function is defined by $\rho^*(\mathcal{S} \setminus A) = |\mathcal{S}| - \rho(\mathcal{S}) - |A| + \rho(A)$, $\forall A \subseteq \mathcal{S}$.*

The function ρ^* is called *corank* function of $\mathcal{M}$. A *cobase* of $\mathcal{M}$ is a base of $\mathcal{M}^*$, a *cocircuit* of $\mathcal{M}$ is a circuit of $\mathcal{M}^*$ and so on.

It is said that a matroid $\mathcal{M}$ is *connected* or *non-separable* if for every pair of distinct elements x and y of $\mathcal{S}$ there is a circuit of $\mathcal{M}$ containing x and y. Moreover, a matroid $\mathcal{M}$ is connected if and only if its dual $\mathcal{M}^*$ is connected. It appears that if a matroid $\mathcal{M}$ is connected then we do not need to know the full set of circuits of $\mathcal{M}$ in order to be able to specify the matroid completely.

Theorem 2. *[17] Let $\mathcal{M}$ be a connected matroid on $\mathcal{S}$ and let x be a fixed element of $\mathcal{S}$. The collection of circuits of $\mathcal{M}$ which contains x uniquely determines $\mathcal{M}$.*

The Tutte polynomial of $\mathcal{M}$ is defined as

$$T(\mathcal{M}, x, y) = \sum_{A \subseteq \mathcal{S}} (x-1)^{\rho(\mathcal{S})-\rho(A)} (y-1)^{|A|-\rho(A)}.$$

The evaluation of this polynomial provides a lot of information about the matroid, e.g., the numbers of bases, the number of independent sets, the number of sets which contain a base, the number of all subsets in $\mathcal{S}$. Moreover, by definition we have that $T(\mathcal{M}, x, y) = T(\mathcal{M}^*, y, x)$.

A matroid is said to be representable over a field $\mathbb{F}_q$ if there exists a vector space V over $\mathbb{F}_q$ together with a map $\phi : S \to V$ which represents the rank. However, the representability problem for matroids is still not completely solved. See, for instance, [17, Chapter 9] for some results on this problem.

2.2 Matroids and Codes

An $[n, k, d]$ linear code $\mathbf{C}$ over a finite field $\mathbb{F}_q$ defines a subspace of dimension k in $\mathbb{F}_q^n$. All codewords have minimum weight d. The dual code $\mathbf{C}^\perp$ consists of elements $\{y : y \cdot x = 0 \text{ for all } x \in \mathbf{C}\}$. Consequently, $\mathbf{C}^\perp$ has parameters $[n, n-k, d^\perp]$, where $d^\perp$ is the minimum distance of the code $\mathbf{C}^\perp$.

A linear code can be defined by two matrices: the generator matrix and the parity check matrix. The generator matrix G of an $[n, k, d]$-code is a $k \times n$-matrix, whose rows form a basis for $\mathbf{C}$. The generator matrix H of $\mathbf{C}^\perp$ is called the parity check matrix of the code, which is an $(n-k) \times n$-matrix. Hence, $x \in \mathbf{C}$ if and only if $xH^T = 0$, (since $GH^T = 0$).

The weight enumerator $W_{\mathcal{C}}(x, y)$ of the code $\mathcal{C}$ is the homogeneous polynomial

$$W_{\mathcal{C}}(x, y) = \sum_{c \in \mathcal{C}} x^{n-wt(c)} y^{wt(c)} = \sum_{i=0}^{n} A_i x^{n-i} y^i,$$

where A_i represents the number of words of weight i in $\mathbf{C}$.

The connection between the weight enumerator $W_{\mathcal{C}^\perp}(x,y)$ of the dual code $C^\perp$ and the weight enumerator of C is as follows:

$$W_{\mathcal{C}^\perp}(x,y) = \frac{1}{|\mathbf{C}|} W_{\mathcal{C}}(x+(q-1)y, x-y).$$

We refer to [12] for more details on linear codes.

The relation between matroids and linear codes over a given field has been studied in [7]. In short, if G is the generator matrix of a linear code $\mathbf{C}$ in $\mathbb{F}_q^n$, then the matroid $\mathcal{M}(\mathbf{C})$ associated with the code $\mathbf{C}$ is the matroid defined over the set of column indices $\{1,\ldots,n\}$ whose independent sets are the linearly independent columns of G.

Theorem 3. *[3, 4] If the matroid $\mathcal{M}$ corresponds to the code $\mathbf{C}$, then the dual matroid $\mathcal{M}^*$ corresponds to the dual code $\mathbf{C}^\perp$.*

Theorem 4. *[3, 4] Let $\mathbf{C}$ be a code over a field with q elements, and $\mathcal{M}$ is the corresponding matroid. Then*

$$W_{\mathbf{C}}(x,y) = y^{n-dim(\mathbf{C})}(x-y)^{dim(\mathbf{C})} T(\mathcal{M}, \frac{x+(q-1)y}{x-y}, \frac{x}{y}).$$

The analogues of *deletion* and *contraction* of a matroid are the operations of *puncturing* and *shortening* a code.

2.3 Secret Sharing Schemes

Define the set of participants in a secret sharing scheme (SSS) by $\mathcal{P} = \{1,\ldots,n\} = \{P_1,\ldots,P_n\}$ and denote the power set of $\mathcal{P}$ by $P(\mathcal{P})$. The set $\Gamma \subseteq P(\mathcal{P})$ is called monotone increasing if for each set A in Γ, each set containing A is also in Γ. Similarly, the set $\Delta \subseteq P(\mathcal{P})$ is called monotone decreasing, if for each set B in Δ each subset of B is also in Δ. A monotone increasing set Γ can be described efficiently by the set Γ^- consisting of the minimal elements (sets) in Γ, i.e., the elements in Γ for which no proper subset is also in Γ. Similarly, the set Δ^+ consists of the maximal elements (sets) in Δ, i.e., the elements in Δ for which no proper superset is also in Δ.

The tuple (Γ, Δ) defines an *access structure* on $\mathcal{P}$ when $\Gamma \cap \Delta = \emptyset$. When $\Delta = P(\mathcal{P}) \setminus \Gamma$ (i.e. $\Delta = \Gamma^c$) then the access structure (Γ, Δ) is said to be *complete* and is denoted just by Γ. If Δ consists of all elements of weight less than k, we call the access structure threshold and denote it by $\Gamma_{k,n}$.

The dual sets Δ^* and Γ^* to Γ and Δ, respectively, are defined by $\Gamma^* = \{A : \mathcal{P} \setminus A \in \Delta\}$ and $\Delta^* = \{A : \mathcal{P} \setminus A \in \Gamma\}$. The tuple (Γ^*, Δ^*) (or Γ^* when it is complete) is called the *dual access structure*. It is easy to see that Δ^* is monotone decreasing and Γ^* is monotone increasing. For two monotone decreasing sets Δ_1 and Δ_2 define $\Delta_1 \uplus \Delta_2 = \{A = A_1 \cup A_2; A_1 \in \Delta_1, A_2 \in \Delta_2\}$. Note that $\Delta_1 \uplus \Delta_2$ is again a monotone decreasing set.

A secret sharing scheme allows the dealer P_0 to share a secret among n participants in such a way that some sets of participants (those in Γ), called allowed

coalitions, can recover the secret, while any other set of participants (non-allowed coalitions) cannot get any information about the secret. The scheme is called *ideal* if the size of any share coincides with the size of the secret. If the share of any participant is computed by a fixed linear function of the key and some other random elements, the SSS is said to be linear (shortly denoted as LSSS).

An access structure is called *ideal* if there is an ideal SSS realizing it. For an access structure (Γ, Δ), $core(\Gamma)$ is defined to be the set of players which are in some minimal qualified set, that is, $core(\Gamma) = \cup_{A \in \Gamma^-} A$. An access structure (Γ, Δ) is called *connected* if $core(\Gamma) = \mathcal{P}$.

3 Error-Set Codes

The linear $[n, k, d]$-code over $\mathbb{F}_q$ can be generalized to the linear $[N, k, \Delta]$-code $\mathbf{C}$ over $\mathbb{F}_q$. The $[N, k, \Delta]$-code is called an error-set code [14] because of the property that all vectors for which the support belongs to Δ are no codewords. It also implies that if x is a codeword then $\sup(x) \notin \Delta$. The set Δ is called the set of forbidden distances and denoted by $\Delta(\mathbf{C})$. In its most general definition for the error-set code, the set Δ is defined over the set (of sub-vectors formed by a partition) $\{1, \ldots, n+1\}$. When instead Δ is defined over the set (of coordinates) $\{1, \ldots, N\}$ the code is called ideal and thus $N = n + 1$. It is clear that for $\Delta = \{A : |A| \leq d - 1\}$, the $[n + 1, k, \Delta]$-code coincides with the usual definition of $[n + 1, k, d]$-code. We will consider further only the ideal error-set codes. Let us first derive some new properties of these codes.

Theorem 5. *The parity check matrix of an $[n + 1, k, \Delta(\mathbf{C})]$-code is a matroid defined on the set of column indices $\mathcal{S} = \{1, \ldots, n + 1\}$ with independent set $\mathcal{I} = \Delta(\mathbf{C})$.*

Proof. A vector x does not belong to the code if and only if $Hx^T \neq 0$. This also means that the columns corresponding to the indices defined by $\sup(x)$ are linearly independent. Recall that by Theorem 1 the columns corresponding to the indices defined by the supports of the vectors from $\Delta(\mathbf{C})$ define the independent sets of a matroid on $\mathcal{S}$. □

Theorem 6. *The dual of an $[n+1, k, \Delta(\mathbf{C})]$-code is an $[n+1, n+1-k, \Delta(\mathbf{C}^\perp)]$-code with $\Delta(\mathbf{C}^\perp) = \Delta(\mathbf{C})^*$.*

Proof. By Theorem 3 the dual of a matroid with independent sets defined by $\Delta(\mathbf{C})$ is the matroid with independent sets defined by $\Delta(\mathbf{C})^*$. This matroid defines the parity check matrix of the dual code. By Theorem 5, the dual code cannot have vectors with support belonging to $\Delta(\mathbf{C}^\perp)$. □

Corollary 1. *The generator matrix of an $[n+1, k, \Delta(\mathbf{C})]$-code is equivalent to a matroid defined on the set $\mathcal{S} = \{1, \ldots, n+1\}$ with an independent set $\mathcal{I} = \Delta(\mathbf{C})^*$.*

Example 1. Consider the $[5, 3, \Delta(\mathcal{C})]$-code with its corresponding dual the $[5, 2, \Delta(\mathcal{C})^*]$-code where $(\Delta(\mathcal{C})^*)^+ = \{\{2, 3, 5\}, \{1, 3, 5\}, \{1, 2, 5\}, \{2, 4, 5\}, \{1, 4, 5\},$

$\{1,2,4\},\{1,3,4\},\{2,3,4\}\}$ and $\Delta(\mathcal{C})^+ = \{\{1,4\},\{2,4\},\{3,4\},\{1,3\},\{2,3\},\{3,5\},\{2,5\},\{1,5\}\}$. The generator matrix G and parity check matrix H of the code are given by:

$$G = \begin{pmatrix} 1\,0\,1\,1\,0 \\ 0\,1\,1\,1\,0 \\ 0\,0\,0\,1\,1 \end{pmatrix}, \quad H = \begin{pmatrix} 1\,1\,1\,0\,0 \\ 0\,0\,1\,1\,1 \end{pmatrix}$$

The linearly independent sets of columns in H correspond to the elements of $\Delta(\mathbf{C})$, while the linearly independent sets of columns in G correspond to the elements of $\Delta(\mathbf{C})^*$.

Another way for determining codes and matroids for a given generator matrix G is given in the following two theorems.

Theorem 7. *Set A belongs to $\Delta(\mathbf{C})^+$ if and only if the matrix G of an $[n+1,k,\Delta(\mathbf{C})]$-code obtained by removing the columns corresponding to A has rank k.*

Theorem 8. *Set A belongs to $\Delta(\mathbf{C}^\perp)^+$ if and only if the columns of the generator matrix G of an $[n+1,k,\Delta(\mathbf{C})]$-code corresponding to A are linearly independent.*

Theorem 9. *Consider the $[n+1,k,\Delta(\mathbf{C})]$-code $\mathcal{C}$ and the corresponding dual $[n+1,n+1-k,\Delta(\mathbf{C}^\perp)]$-code $C^\perp$. The elements of $\Delta(\mathbf{C})^+$ have size $(n+1-k)$ and the elements of $\Delta(\mathbf{C}^\perp)$ have size k.*

Proof. The size of the elements of $\Delta(\mathbf{C})^+$ (resp. $\Delta(\mathbf{C}^\perp)^+$) follows from the rank of matrix H (resp. G). □

By Corollary 1 and by Theorem 4 we can derive the weight enumerator of an $[n+1,k,\Delta(\mathbf{C})]$-code.

Theorem 10. *[7] Let C be an $[n+1,k,\Delta(\mathbf{C})]$-code and $\mathcal{M}$ be the matroid with independent sets defined by $\Delta(\mathbf{C})^*$. Then*

$$W_{\mathcal{C}}(x,y) = y^{n+1-r(\mathcal{M})}(x-y)^{r(\mathcal{M})}T(M;\frac{x+(q-1)y}{x-y},\frac{x}{y}). \tag{1}$$

4 Relations Between Matroids, Codes and LSSS

The following equivalence relations between matroids and ideal LSSS on one hand and LSSS and error-set codes on the other hand are known. First we briefly recall these equivalences and using the results from the previous section we close the chain of relations by establishing directly the equivalence between ideal error-set codes and matroids.

4.1 LSSS and Matroids

Brickell and Davenport were the first to point out the relation between ideal LSSS and matroids over the set $S = \{0, 1, \ldots, n\} = \{P_0\} \cup \mathcal{P}$.

Definition 2. *[1] Let M be an ideal SSS, The Δ-set of such a scheme is*

$$D(M) = \{A \subseteq \mathcal{P} : \exists y \in A \text{ such that } y \sim A \setminus y\}.$$

Theorem 11. *[1, 6, 9] Let Γ be an ideal connected access structure on $\mathcal{P}$ with $\Gamma^- = \{C_i, i \in I\}$. Then the sets $\{P_0\} \cup C_i$, $i \in I$ are all circuits through P_0 of a unique matroid (by Theorem 2) defined on $\mathcal{S} = \{P_0\} \cup \mathcal{P}$. This matroid is called to be induced by Γ, and denoted by $\mathcal{M}(\Gamma)$. The sets $D(M)$ are the dependent sets of the connected matroid.*

Theorem 12. *[15] Let Γ be an ideal access structure for the ideal SSS M. The sets $X \in \Gamma^-$ and sets $(X \setminus \{P_i\}) \cup \{P_0\}$ for $P_i \in X$, $X \in \Gamma^-$ form the bases of a representable matroid $\mathcal{M}(\Gamma)$ if and only if Γ and M satisfy the requirement $X \in \Gamma^- \Leftrightarrow$ rows of M_X are independent.*

The opposite relation is also true.

Theorem 13. *[1] Let $\mathcal{M} = (\mathcal{S}, \mathcal{I})$ be a connected representable matroid and let $P_0 \in \mathcal{S}$. Then there exists a connected ideal SSS M on $\mathcal{P} = \mathcal{S} \setminus \{P_0\}$ with a dealer P_0 and a target vector ε and such that $D(M) = \mathcal{I}$.*

A relation between the dual access structures and matroids also holds.

Theorem 14. *[6, 9] Let Γ be a connected ideal access structure that induces a matroid $\mathcal{M}(\Gamma)$. Then Γ^* induces a matroid $\mathcal{M}(\Gamma^*)$ and*

$$\mathcal{M}(\Gamma)^* = \mathcal{M}(\Gamma^*).$$

Theorem 15. *[6, 9] Let Γ be an ideal access structure that induces a matroid $\mathcal{M}(\Gamma)$. Then Γ is connected if and only if $\mathcal{M}(\Gamma)$ is connected.*

4.2 LSSS and Codes

The connection between LSSS and error-set codes is made using the concept of MSP.

Definition 3. *[10] A Monotone Span Program (MSP) $\mathcal{M}$ is defined by the quadruple $(\mathbb{F}, M, \varepsilon, \psi)$, where $\mathbb{F}$ is a finite field, M is a matrix (with m rows and $d \leq m$ columns) over $\mathbb{F}$, $\psi : \{1, \ldots, m\} \to \{1, \ldots, n\}$ is a surjective functions and $\varepsilon = (1, 0, \ldots, 0)$ is a fixed non-zero vector, called target vector. The function ψ labels each row with a number i that corresponds to player $P_{\psi(i)}$. The size of $\mathcal{M}$ is the number of rows and is denoted as size($\mathcal{M}$).*

Let M be an $m \times d$ matrix and let M_A be the matrix consisting of the rows owned by A. An MSP is said to compute the access structure Γ when $\varepsilon \in M_A^T$ if and only if $A \in \Gamma$. Or equivalently

$$A \in \Gamma \Leftrightarrow \exists \lambda \in \mathbb{F}^{|A|} \text{ such that } M_A^T \lambda = \varepsilon$$
$$B \notin \Gamma \Leftrightarrow \exists k \in \mathbb{F}^d \text{ such that } M_B k = 0 \text{ and } k_1 = 1.$$

The MSP which computes Γ^* is called the dual MSP $\mathcal{M}^*$ with corresponding matrix M^*. A relation between dual access structures and dual codes has been established as follows.

Theorem 16. *[13] Let $\mathcal{M}$ be an MSP program computing Γ, and $\mathcal{M}^\perp$ be an MSP computing the dual access structure $\Gamma^\perp$. Let code $\mathcal{C}^\perp$ have the parity check matrix $H^\perp = (\boldsymbol{\varepsilon} \mid (M^\perp)^T)$ and let code $\mathcal{C}$ have the parity check matrix $H = (\boldsymbol{\varepsilon} \mid M^T)$. Then for any MSP $\mathcal{M}$ there exists an MSP $\mathcal{M}^\perp$, such that $\mathcal{C}$ and $\mathcal{C}^\perp$ are dual.*

For expressing the relations between LSSS and error-set codes, we need to work with a particular type of MSP.

Definition 4. *[13] An MSP $\mathcal{M} = (\mathbb{F}, M, \boldsymbol{\varepsilon}, \psi)$ is called a Δ-*non-redundant monotone span program *(denoted by Δ-rMSP), if $\ker(M_A^T) = \{0\}$ holds for any $A \in \Delta$.*

Theorem 17. *[14] Consider the Δ-rMSP $\mathcal{M}$. Let $\mathbf{C}$ be an error-set correcting code with a generator matrix G of the form $G = (\boldsymbol{\varepsilon} \mid M^T)$. Then $\mathbf{C}$ defines an LSSS with the set of forbidden distances $\Delta(\mathbf{C})$ equal to $\Delta^\perp \uplus \{\emptyset, \{P_0\}\}$.*

4.3 Codes and Matroids

For the threshold case, all relations are well known: every threshold access structure $\Gamma_{k,n}$ realized by an SSS over $\mathbb{F}_q$ has an associate uniform matroid $\mathcal{M}_{k,n+1}$ and corresponds to an MDS $[n+1, k, n-k+2]$-code. Thus $\mathcal{M}_{k,n+1} = \mathcal{M}(\Gamma_{k,n}) = \mathcal{M}([n+1, k, n-k+2])$ hold. It seems that similar relations hold for general access structures.

Let us analyze the equivalence between Theorem 12 and Theorem 17 for the ideal case. The definition of Δ-rMSP corresponds to the property that $X \in \Gamma^-$ if and only if the rows of M_X are linearly independent. Furthermore, Theorem 5 where the relation between matroids and error-set codes is expressed, shows the equivalence between the assumptions of both theorems.

For a relation between the connected LSSS and the error-set codes, it is clear by Theorem 5 that the only extra requirement in Theorem 17 will be that the set $\Delta(\mathbf{C})$ satisfies the properties of a connected matroid. Moreover, the equivalence between Theorem 14 and Theorem 16 follows from Theorem 17.

5 Conclusion

We continued the study of combinatorial objects defined in a setting where the set of positions in which two vectors differ is used as a metric and the support

of a vector as a norm. More precisely, we have shown the relation between the $[N, k, \Delta]$-error-set codes and the matroids.

References

1. E. Brickell and D. Davenport. On the classification of ideal secret sharing schemes, *J. of Cryptology* 4, 1991, pp. 123–134.
2. A. Braeken, V. Nikov, S. Nikova, B. Preneel. On Boolean Functions with Generalized Cryptographic Properties, *INDOCRYPT'04*, LNCS 3348, 2004, pp. 119-134. full version - *Cryptology ePrint Archive*: Report 2004/259.
3. P. Cameron. Codes, Matroids and Trelises, preprint.
4. P. Cameron. Polynomial aspects of codes, matroids and permutation groups, *Lecture Notes*, 2002.
5. M.van Dijk. Secret Key Sharing and Secret Key Generation, *Ph.D. Thesis*, TU Eindhoven, 1997.
6. M.van Dijk, W.-A. Jackson and K. Martin. A note on duality in linear secret sharing scheme, *Bulletin of the Institute of Combinatorics and its Application*, 19, 1997, pp. 93–101.
7. C. Greene. Weight Enumerator and the Geometry of Linear Codes, *Studies in Applied Mathematics*, 55, 1976, pp. 119–128.
8. K. Luders-Jensen. Secret Sharing, *Master Thesis, Dept. of Math. DTU*, 1996.
9. W.-A. Jackson and K. Martin. Geometric Secret Sharing Schemes and Their Duals, *Designs Codes and Cryptography*, 4, 1994, pp. 83–95.
10. M. Karchmer and A. Wigderson. On Span Programs, *Proc. of 8th Annual Structure in Complexity Theory Conference*, 1993, pp. 102-111.
11. J. Massey. Minimal codewords and secret sharing, *Proc. 6th Joint Swedish-Russian Int. Workshop on Inform. Theory*, 1993, pp. 276-279.
12. F.J. MacWilliams and N.J.A. Sloane. The Theory of Error-Correcting Codes, *Elsevier*, 1991.
13. V. Nikov, S. Nikova, B. Preneel. On the size of Monotone Span Programs, *SCN'04*, LNCS 3352, 2004, pp. 252-265.
14. V. Nikov, S. Nikova. On a relation between Verifiable Secret Sharing Schemes and a class of Error-Correcting Codes, *Cryptology ePrint Archive:* Report 2003/210.
15. T. Uehara, T. Nishizeki, T. Okamoto and K. Nakamura. A Secret Sharing System with Matroid Access Structure, *Trans. IECE Japan*, J69-A **9**, 1986, pp. 1124–1132.
16. H. Whitney. On the Abstract Properties of Linear Dependence, *American Journal of Mathematics*, 57, 1935, pp. 509–533.
17. D. Welsh. Matroid Theory, *Academic Press, London*, 1976.

On Walrasian Price of CPU Time
Extended Abstract

Xiaotie Deng[1,⋆], Li-Sha Huang[2,⋆⋆], and Minming Li[2]

[1] Department of Computer Science, City Univ. of Hong Kong
csdeng@cityu.edu.hk
[2] State Key Laboratory of Intelligent Technology and Systems
Dept. of Computer Science and Technology, Tsinghua Univ., Beijing, China
{hs02,liminming98}@mails.tsinghua.edu.cn

Abstract. We study a Walrasian Equilibrium model to determine the prices of CPU time as merchandise. The customers have jobs that require a given length of CPU slot allocation with their valuations dependent on the assigned time slots. The owner of CPU processing time receives compensation for time slots sold to the customers, subject to the condition that the slots sold to a customer is those that customer most desires, given the price structure for the time slots. We establish conditions for jobs to have Walrasian Equilibrium, and obtain algorithm and complexity results to determine Walrasian equilibrium price and allocation. In particular, the issues of excessive supply of CPU time and price dynamics are discussed under our model.

1 Introduction

Information technology has changed our lifestyles with the creation of many new consumer products, such as word processing software, computer games, search engines, and online communities. Digital goods and services are fast becoming everyone's shopping items. While the new goods of Information Age are enriching the market place with ever-changing products, they pose a great challenge to our understanding of economics. Such a new economy has already demanded many theoretical tools (new and old, of economics and other related disciplines) be applied to their development and production, marketing and pricing (see e.g. [10]). At the same time, no theory have been able to paint a clear picture of the reality, far less so in comparison with classic economics.

The lack of a full understanding of the new economy is mainly due to the fact that digital goods can often be re-produced at no additional cost, though multi-fold other factors could also be part of the difficulty. Not surprisingly, the marketplace practice of digital goods is anything but what is predicted by classic economics for commodities:

⋆ Supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 1156/04E).
⋆⋆ Supported by Natural Science Foundation of China (No.60135010,60321002) and the Chinese National Key Foundation Research and Development Plan (2004CB318108).

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 586–595, 2005.

- While the price is much influenced by the marginal cost re-production, digital goods are not all sold at zero price but a wide range of possible (and seemingly arbitrary) prices.
- While consumers of the same product pay the same price in classic economics, differentiated pricing is a common practice for digital goods.
- While all positive priced goods are cleared in classic economics, digital goods and services with excessive supplies are often not free.

It is understandable to have some of such contradictions to classic economics because of the special characteristics of digital goods, pointed out above. It may well be treated as a special extreme case for the commodity economy. However, as the Internet economy becomes an indispensable part of everyone's life, it is unavoidable that one may quest for a comprehensive theoretical understanding of digital goods pricing mechanism. In this work, we take a humble step on such a mission by focusing on CPU time as a product for sale in the market. We study it with the Walrasian pricing model in economics.

CPU time as a commercial product is extensively studied in grid computing (see, e.g., [8]). Singling out CPU time pricing will help us to set aside other complicated issues caused by secondary factors, and a complete understanding of this special digital product (or service) may shed some light on the study of other goods in the digital economy.

The utilization of CPU time by multiple customers has been a crucial issue in the development of operating system concept. The rise of grid computing proposes to fully utilize computational resources, e.g. CPU time, disk space, bandwidth. Market oriented schemes have been proposed for efficient allocation of computational grid recourses, by [9, 12]. And later, various practical and simulation systems have emerged in grid resource management, e.g., Spawn [15], Popcorn [14], D'Agents [3], etc.. Besides the resource allocation in grids, Feigenbaum *et. al.* [7] gave an example of introducing economic mechanism into routing between Internet domains.

Our approach deals with the relationship of key concepts in the economy: commodity, price, and customer valuation. We are interested in the price equilibrium model in the tradition of Walras [16], Arrow and Debreu [1] and the complexity of computing equilibrium [6]. In most of CPU allocation models, CPU time is treated as the same commodity that would reduce the theoretical problem to the classic model of one commodity economy, and the rigidity of customer job makes it further simplified. For such customer jobs, price equilibrium is quite impossible since the CPU time is often not fully utilized. The equilibrium price would be, in such case of excessive supply, zero. We explore a more general job model that the customer valuation would be dependent on the completion time of its job. We also study the non-increasing propert of price sequence in job scheduling models, which can be viewed as a comparative work to Chen *et. al.* [5]'s study in the price sequence of online auctions.

The paper is organized as follows. In Section 2, the relevant definitions and necessary preliminaries are introduced. In Section 3, we first establish a theorem on the existence of Walrasian Equilibrium for our general CPU job model, in the

traditional linear programming and integer programming paradigm. With this theorem, we prove that it is NP-hard to determine the existence of Walrasian Equilibrium in job scheduling model. In Section 4, we focus on a class of linear valuation functions. As a positive result, we prove that Walrasian Equilibrium exists in the model if the total available CPU time is a little more than the sum of all required CPU service time. In Section 5, we establishes a non-increasing property of equilibrium price for job scheduling models. We conclude our work in Section 6 with remarks and discussion on the current results and possible future extensions.

2 Preliminaries

In this section, we introduce the job scheduling problem, a model of valuation functions, and Walrasian Equilibrium price.

2.1 XOR Bids and Valuation Functions

We adopt the notion of combinatorial auctions [13] in our discussion, which is helpful for us to establish our results. Consider an exchange economy (Ω, I, V):

- Commodities: The seller sells m kinds of indivisible commodities in the market. Let $\Omega = \{\omega_1 \times \delta_1, ..., \omega_m \times \delta_m\}$ denote the set of commodities where δ_j is the available quantity of the item ω_j.
- Agents: There are n agents in the market acting as buyers, denoted by $I = \{1, 2, ..., n\}$.
- Valuation functions: Each buyer $i \in I$ has a valuation function $v_i : 2^{\Omega} \to \mathbb{R}^+$ to submit the maximum amount of money he is willing to pay for a certain bundle of items. $V = \{v_1, v_2, ..., v_n\}$.

Nisan [13] introduced a formalism, the bidding language, to express various valuation functions. Any valuation function can be presented in this form [13].

Definition 1. *[13] An XOR combination of two valuation functions v_1 and v_2 is defined by:*

$$(v_1 \ XOR \ v_2)(S) = \max\{v_1(S), v_2(S)\}$$

An atomic bid, or so called single-minded bid, is a valuation function v defined by a pair $\{S, q\}$ where $S \subset \Omega$ and $q \in \mathbb{R}^+$:

$$v(T) = \begin{cases} q & \text{if } S \subset T \\ 0 & \text{otherwise} \end{cases}$$

An XOR bid is a combination of several atomic bids by XOR operators, written as

$$v = (S_1, q_1) \ XOR \ (S_2, q_2) ... \ XOR \ (S_n, q_n)$$

Given (Ω, I, V) as input, the market is to determine an *allocation* and a *price vector* as output:

- An *allocation* $X = \{X_0, X_1, X_2, ..., X_n\}$ is a partition of Ω in which X_i is the bundle of commodities assigned to buyer i and X_0 is the set of unallocated commodities.
- A *price* vector p is a nonnegative vector in $\mathbb{R}^m$ whose j-th entry is the price of good $\omega_j \in \Omega$.

The *social efficiency* of an allocation X is the sum of all buyers' valuation: $\sum_{i=1}^{n} v_i(X_i)$. An allocation $X^* = \{X_0^*, X_1^*, ..., X_n^*\}$ is said to be *optimal* if it maximizes social efficiency, i.e. $\sum_{i=1}^{n} v_i(X_i^*) \geq \sum_{i=1}^{n} v_i(X_i)$ for any other allocation $X = \{X_0, X_1, ..., X_n\}$.

For any subset $T = \{\omega_1 \times \sigma_1, ..., \omega_m \times \sigma_m\} \subset \Omega$, define $p(T)$ as $p(T) = \sum_{j=1}^{m} \sigma_j p_j$. If buyer i is assigned to a bundle of commodities X_i and the price vector is p, his *utility* is defined to be $u_i(X_i, p) = v_i(X_i) - p(X_i)$.

2.2 The CPU Job Scheduling Problem

We consider two types of players in a market-driven CPU resource allocation model: a resource provider and n consumers. The provider sells to the consumers CPU time slots and the consumers each has a job that requires a fixed number of CPU time, and its valuation function depends on the time slots assigned to the job, usually the last assigned CPU time slot. We assume that all jobs are released at time $t = 0$ and the i-th job needs s_i time units. We denote by $v_i(\cdot)$ the valuation function of agent i on the time slots assigned to it. In general, the jobs may be or may not be interruptible but we focus on jobs that are interruptible without preemption cost, as is often modelled for CPU jobs.

Using the notion of the previous subsection, for the job scheduling problem, there are m commodities (time units), $\Omega = \{\omega_1, ..., \omega_m\}$, and n buyers (jobs) , $I = \{1, 2, ..., n\}$, in the market. Each buyer has a valuation function v_i, usually only dependent on the completion time. Moreover, if not explicitly mentioned, every job's valuation function is non-increasing w.r.t. completion time. We call such valuation functions *non-increasing valuation functions*.

2.3 Walrasian Equilibrium Price

Definition 2. *[11]A Walrasian Equilibrium for an exchange economy (Ω, I, V) is a tuple (X, p), where $X = \{X_0, X_1, ..., X_n\}$ is an allocation and p is a price vector, satisfying that:*

$$(1)\ p(X_0) = 0;$$
$$(2)\ u_i(X_i, p) \geq u_i(B, p), \forall B \subset \Omega, \forall 1 \leq i \leq n$$

Such a price vector is also called a market clearing price, or Walrasian price, or equilibrium price.

There is a well-known proposition of Walrasian equilibrium:

Proposition 1. *[4] Walrasian equilibrium maximizes the social efficiency, i.e. if (X, p) is a Walrasian equilibrium, then X is an optimal allocation.*

Example 1. Two non-interruptible jobs compete for four time units $\{\omega_1, \omega_2, \omega_3, \omega_4\}$. Their valuation functions are:

$$v_1 = (\{\omega_1, \omega_2\}, 7) \text{ XOR } (\{\omega_2, \omega_3\}, 4) \text{ XOR } (\{\omega_3, \omega_4\}, 1)$$
$$v_2 = (\{\omega_1\}, 7) \text{ XOR } (\{\omega_2\}, 5) \text{ XOR } (\{\omega_3\}, 3) \text{ XOR } (\{\omega_4\}, 1)$$

The equilibrium price is $(3, 1, 0, 0)$. Job 1 gets $\{\omega_2, \omega_3\}$, job 2 gets $\{\omega_1\}$. It is an example of Walrasian Equilibrium for the job scheduling problem.

3 Existence and Complexity

In this section, we propose a sufficient and necessary condition for the existence of Walrasian Equilibrium in an exchange economy with indivisible commodities. As its application, we show that deciding the existence of Walrasian Equilibrium is strong NP-hard even when restricted to job scheduling models.

3.1 Relation to Linear Programming

Bikhchandani *et. al.* [2] proved that Walrasian Equilibrium exists in an exchange economy with indivisible commodities if and only if the buyers' welfare cannot be improved by making the commodities divisible. In other words, Bikhchandani's theorem [2] claims that Walrasian Equilibrium exists if and only if the optimum of an integer programming problem equals the optimum of its linear relaxation.

However, the size of their linear programming problem is exponential to the total number of commodities which is unacceptable computationally. Chen *et. al.* [4] obtained a similar result for atomic bids while the complexity is linear to the number of commodities and buyers. We extend their result to XOR bids and show that the size of the linear programming problem is linear to the number of items and XOR clauses.

Assume in an economy, $\Omega = \{\omega_1 \times \delta_1, ..., \omega_m \times \delta_m\}$ is the set of commodities, $I = \{1, 2, ..., n\}$ is the set of buyers. Each buyer i has a valuation function which can be represented by r_i pairs:

$$(S_{i1}, q_{i1}) \text{ XOR } (S_{i2}, q_{i2}) \text{ XOR } ... \text{ XOR } (S_{ir_i}, q_{ir_i})$$

Maximization of social efficiency is equivalent to solving the following linear programming problem when items are divisible:

LPR:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \sum_{j=1}^{k_i} q_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{i,j|\omega_k \in S_{ij}} x_{ij} \leq \delta_k, \forall \omega_k \in \Omega \\ & \sum_{j=1}^{r_i} x_{ij} \leq 1, \forall 1 \leq i \leq n \\ & 0 \leq x_{ij} \leq 1, \forall i, j \end{aligned}$$

Denote its integer restriction as **IP**. Now we can reach the main theorem of this subsection:

Theorem 1. *In an economy with indivisible commodities and XOR valuation functions, the Walrasian Equilibrium exists if and only if the optimum of **IP** equals the optimum of **LPR**. The size of LP problem is linear to the total number of XOR bids.*

The proof of this theorem can be found in the full version of this paper.

3.2 Reducing LP Size for Non-increasing Valuation Function

The main difficulty of directly applying Theorem 1 to job scheduling problems is that the number of XOR clauses is exponential to the number of available time units if the jobs are allowed to be interrupted. More precisely, if the number of total time units is m and job i's time span is s_i, then there are $\binom{m}{s_i}$ XOR clauses in the valuation function v_i. In this subsection, we try to overcome this obstacle.

In a feasible allocation, the allocated time units to every job must be or may not be consecutive. We call the former one *consecutive allocation*, and the latter *general allocation*.

Lemma 1. *General allocation can not gain more social efficiency than consecutive allocation.*

Lemma 1 shows that general allocation cannot gain more social efficiency than consecutive allocation, when the valuation functions are non-increasing w.r.t completion time. What happens if we stands on Walrasian Equilibrium's point of view?

In the next theorem, if jobs are all interruptible, we call the scheduling problem *general scheduling*, and *restricted scheduling* if jobs are all non-interruptible.

Theorem 2. *Walrasian Equilibrium exists in the general scheduling model if and only if Walrasian Equilibrium exists in the restricted scheduling version.*

Compared to $\binom{m}{s_i}$ clauses in one valuation function of general scheduling problem, there are only $(m - s_i + 1)$ XOR clauses in the restricted version. Hence Theorem 2 dramatically reduces the scale of the corresponding linear programming problem in Theorem 1. The following example illustrates an application of Theorem 1 and Theorem 2.

Example 2. Suppose there are three time units for two buyers.

$$v_1 = (\{1,2\}, 15) \text{ XOR } (\{2,3\}, 2)$$
$$v_2 = (\{1\}, 20) \text{ XOR } (\{2\}, 20) \text{ XOR } (\{3\}, 8)$$

It is easy to see that the optimal integer allocation produces the social efficiency 23. However, the linear program can yield a better solution 27.5 by distributing $\{1,2\} \times 0.5$ to buyer 1 and $\{1\} \times 0.5 + \{2\} \times 0.5$ to buyer 2. Therefore, Walrasian Equilibrium does not exist in the example.

3.3 Strong NP-Hardness

Although Theorem 1 can help us to determine the existence of Walrasian Equilibrium in job scheduling problems, it is still a hard problem because integer programming is hard. In this subsection, we will show that it is strong NP-hard to decide whether Walrasian Equilibrium exists in a job scheduling problem.

DWE Problem: Given m time units and altogether n jobs. The i-th job needs s_i time units. Each job's valuation on its allocated time units only depends on its completion time and is non-increasing w.r.t. this time. Determine the existence of Walrasian Equilibrium in this job scheduling problem.

We will reduce a strong NP-hard problem, 3-Partition, to DWE problem.

3-Partition problem: Given a set of $3N$ integer numbers $S = \{s_1, s_2, ...s_{3N}\}$, and an integer B, which satisfy $\sum_{i=1}^{3N} s_i = NB$, and $B/4 < s_i < B/2$. Determine whether there exists a partition of S into $P_1, P_2, ...P_N$, such that

$$\sum_{j \in P_i} s_j = B \text{ for all } 1 \le i \le N \tag{1}$$

For the preceding 3-Partition problem, we construct a DWE problem in which there are altogether NB time units, with $3N$ jobs each applying for s_i time units respectively. If job i's completion time is t_i, its valuation is set to be $s_i(N - \lceil \frac{t_i}{B} \rceil + 1)$.

This job scheduling problem naturally induces an integer programming problem and its linear relaxation. Denote the optimum of these two programming problems by M_{IP} and M_{LP}. By Theorem 1, $M_{IP} = M_{LP}$ if and only if DWE admits a Walrasian Equilibrium.

Lemma 2. $M_{LP} \le BN(N+1)/2$.

Lemma 3. *3-Partition has a solution if and only if the maximal social efficiency in the corresponding job scheduling problem equals* $BN(N+1)/2$.

Theorem 3. *Determination of Walrasian Equilibrium Existence(DWE) is strong NP-hard.*

Proof. With an oracle of DWE problem, we can solve 3-Partition in polynomial time:

If the oracle declares that Walrasian Equilibrium doesn't exist in the job scheduling problem, then due to Theorem 1 and Lemma 2, $M_{IP} < M_{LP} \leq BN(N+1)/2$ which means that the social efficiency of the best allocation is less than $BN(N+1)/2$. Thus Lemma 3 ensures that 3-Partition does not have a solution.

If the oracle declares that there exists a Walrasian Equilibrium, due to Theorem 1, we can in polynomial time obtain the maximal social efficiency by solving the linear programming problem. A simple comparison will yield whether 3-Partition has a solution.

4 Excessive CPU Time and Equilibrium in MWCT Model

In the section, we concentrate on a scheduling problem with linear valuation functions. Assume n jobs are released at the time $t = 1$ for a single machine, the j-th job's time span is $s_j \in \mathbb{N}^+$ and weight $w_j \geq 0$. The goal of the scheduling is to minimize the weighted completion time: $\sum_{i=1}^{n} w_i t_i$, where t_i is the completion time of job i. Such a problem is called MWCT (Minimal Weighted Completion Time) in this section.

We can convert an MWCT problem to an exchange economy: the market sells m commodities $T = \{t_1, ..., t_m\}$ (time slots) to n buyers $I = \{1, 2, ..., n\}$ (jobs). The valuation of buyer i to a bundle T_i is $v_i(T_i) = w_i(m - t)$, where $|T_i| = s_i$ and t is the largest item in T_i (completion time). Due to the nice form of the valuation functions, we immediately have the proposition:

Proposition 2. *The social optimum in the economy is equivalent to the optimum in its corresponding MWCT problem.*

By Theorem 2, we can w.o.l.g. assume that buyer i only applies for consecutive time units. Note that there is a classical $O(n \log n)$-time algorithm to find the optimum of MWCT when $m \geq \sum_{i=1}^{n} s_i$. It just simply executes the jobs in a *heavier average weight earlier* order, i.e. if $w_i/s_i > w_j/s_j$, then job i must be executed before job j.

Though the universal problem for MWCT is intractable both at optimization and determination of existence of equilibrium, we do have some promising result when the total number of processor time is large enough:

Theorem 4. *In a single machine MWCT job scheduling problem, Walrasian Equilibrium always exists when $m \geq EM + \Delta$, where m is the total number of processor time, $EM = \sum_{i=1}^{n} s_i$ and $\Delta = \max_k \{s_k\}$.*

The following example shows that the enough excessive CPU time is necessary.

Example 3. There are two jobs $\{1, 2\}$ and five CPU time slots $\{t_1, t_2, t_3, t_4, t_5\}$ in the market. $w_1 = 3, s_1 = 2; w_2 = 4, s_2 = 3$. Allocate (t_1, t_2) to job 1 and (t_3, t_4, t_5) to job 2 will produce social efficiency at 9. However, allocate $(t_1, t_2) \times 0.5$ and $(t_3, t_4) \times 0.5$ to job 1 and $(t_1, t_2, t_3) \times 0.5$ to job 2 will produce social efficiency at 10. Hence by Theorem 1, the Walrasian Equilibrium does not exist.

5 Price Sequence

If Walrasian Equilibrium exists, then we can find not only an optimal schedule but also a price vector for all time units. In this section, we prove the existence of non-increasing price sequence if Walrasian Equilibrium exists under the assumption that the valuation functions are non-increasing w.r.t. completion time.

The proofs of the following lemmas can be found in the full version of this paper.

Lemma 4. *If Walrasian Equilibrium exists with a general allocation and an equilibrium price, then there exists a Walrasian Equilibrium with a consecutive allocation and corresponding equilibrium price.*

Lemma 5. *Equilibrium price in a consecutive allocation must satisfy the following property: if buyer i is allocated before buyer i', then the price of any time unit allocated to i will be higher than or equal to the price of every time unit allocated to buyer i'.*

Definition 3. *Given two sequences $P = \{p_1, p_2, ...p_m\}$, $Q = \{q_s, q_{s+1}, ...q_m\}$ $(0 < s \leq m)$, define the Minimum s- Sum of P at position i as:*

$$ms_i(P) = \min_{T \subset |i|, |T|=s} \left\{ \sum_{k \in T} p_k \right\}$$

and the Maximal Difference of Q to P as:

$$MD(Q, P) = \max_{s \leq i \leq m} \{q_i - ms_i(P)\}$$

Lemma 6. *For a permutation $\pi : |m| \to |m|$ and a vector $P = (p_1, ..., p_m) \in \mathbb{R}^m$, define $\pi(P) = (p_{\pi(1)}, ..., p_{\pi(m)})$. Given two non-increasing sequences $P = \{p_1, p_2, ...p_m\}$ and $Q = \{q_s, q_{s+1}, ...q_m\}$, we have $MD(Q, P) \leq MD(Q, \pi(P))$.*

Lemma 7. *If Walrasian Equilibrium exists with consecutive allocation, then for every consecutive segment, sorting the prices in non-increasing order will still yield an equilibrium price sequence.*

By Lemma 4, Lemma 5 and Lemma 7, we reach the main theorem of the section:

Theorem 5. *If there exists a Walrasian Equilibrium in a job scheduling problem, we can always let it be an equilibrium with consecutive allocation and a non-increasing equilibrium price vector.*

6 Conclusion and Discussion

In the paper, we have shown in Theorem 1 the relation of the duality theory of linear programming and the existence of Walrasian Equilibrium with indivisible commodities. Theorem 2 and Theorem 3 are the examples of its direct application in algorithmic complexity issues. With similar technique in the proof of Theorem 3, we prove the NP-hardness of determining existence of Walrasian Equilibrium in various job scheduling models.

In Section 4, we study MWCT model and show that enough excessive CPU time always admit Walrasian Equilibrium. We believe that the equilibrium price will lead to an incentive compatible pricing mechanism in the model.

References

1. K. J. Arrow and G. Debreu. Existence of an equilibrium for competitive economy. *Econometrica*, 22:265–290, 1954.
2. S. Bikhchandani and J. W. Mamer. Competitive equalibrium in an exchange economy with indivisibilities. *Journal of Economy Theory*, 74:385–413, 1997.
3. J. Bredin, D. Kotz, and D. Rus. Market-based resource control for mobile agents. In *the Proceeding of the 2nd International Conference on Autonomous Agents*. ACM Press, 1998.
4. Ning Chen, Xiaotie Deng, and Xiaoming Sun. On complexity of single-minded auction. *Journal of Computer and System Sciences*, 69(4):675–687, 2004.
5. Ning Chen, Xiaotie Deng, Xiaoming Sun, and Andrew C. Yao. Dynamic price sequence and incentive compatibility. In *the Proceedings of ICALP*, 2004.
6. X. Deng, C. Papadimitriou, and S. Safra. On the complexity of price equilibria. *Journal of Computer and System Sciences*, 67(2):311–324, 2003.
7. J. Feigenbaum, C. H. Papadimitriou, R. Sami, and S. Shenker. A BGP-based mechanism for lowest-cost routing. In *the Proceedings of PODC*, 2002.
8. D. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. Economic models for allocating resources in computer systems. 1996.
9. D. Ferguson, Y. Yemini, and C. Nikolaou. Microeconomic algorithms for load balancing in distributed computer systems. In *the Proceedings of DCS*, 1988.
10. A. V. Goldberg, J. D. Hartline, and A. Wright. Competitive auctions and digital goods. 2001.
11. F. Gul and E. Stacchetti. Walrasian equilibrium with gross substitutes. *Journal of Economy Theory*, 87:95–124, 1999.
12. J. F. Kurose and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Trans. on Computers*, 38(5):705–717, 1989.
13. N. Nisan. Bidding and allocation in combinatorial auctions. In *ACM Conference on Electronic Commerce*, pages 1–12, 2000.
14. N. Nisan, L. London, O. Regev, and N. Camiel. Globally distributed computation over the internet - the POPCORN project. In *International Conference on Distributed Computing Systems*, 1998.
15. C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, 1992.
16. L. Walras. *Elements d'Economique Politique Pure*. Corbaz, Lausanne, Switzerland, 1874.

On-Line Algorithms for Market Equilibria

Spyros Angelopoulos[1], Atish Das Sarma[2],
Avner Magen[3], and Anastasios Viglas[4]

[1] University of Waterloo
sangelop@cs.uwaterloo.ca
[2] IIT Bombay
atish@cse.iitb.ac.in
[3] University of Toronto
avner@cs.toronto.edu
[4] University of Sydney
tasos@it.usyd.edu.au*

Abstract. We consider a variation of the classical problem of finding prices which guarantee equilibrium in linear markets consisting of divisible goods and agents with money. Specifically, we consider *on-line* algorithms for this problem in which goods are considered on-line, and each good is assigned an irrevocable price. Since exact equilibria will not be found in such a setting, we appeal to the concept of approximate equilibrium defined in previous studies of the problem, to characterize the quality of our solutions. We consider both deterministic and randomized algorithms for finding approximate equilibria. We prove a tight bound on the performance of deterministic algorithms, and show that under certain natural conditions, randomized algorithms lead to market prices which are closer to equilibrium.

1 Introduction

The existence of equilibria in markets is a central problem in mathematical economics, and has attracted enormous interest since the pioneering work of Walras [10] and Fisher [3]. The problem is the following. Consider a market which consists of buyers, each with a certain amount of money, and divisible goods of a certain amount each. The desirability of goods to each of the buyers is expressed by the utility functions of the buyers. The goal is to assign prices to goods such that the buyers can buy their individually optimal bundle of goods in terms of the utility they get and there is no surplus of goods, i.e, the market clears. The theorem of Arrow and Debreu [1] established the existence of equilibrium prices in a very general setting. The proof is nonconstructive, yet recently polynomial-time algorithms for the case of linear utility functions have been presented. These algorithms assume complete knowledge of the entire market meaning that the utility functions of all goods are known beforehand.

* Research supported in part by the European Social Fund (75%) and the Greek Ministry of Education (25%) through a grant of the Operational Program "Pythagoras".

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 596–607, 2005.

This might be the case in a static market, but it does not reflect the situation in a *dynamic* market, in which little, if anything, is known about goods that will appear in the market in the future.

This work focuses on the performance of on-line algorithms for computing or predicting equilibrium prices. The online algorithm assigns prices incrementally trying to approximate the actual equilibrium prices that correspond to the actual offline market problem. Prices are assigned without complete knowledge of the market and cannot change. In a more general setting we can formulate the restriction on re-assigning prices by associating a change in a previously assigned price of a good with a cost which measures how undesirable such a change would be. The online setting addressed in this work captures the special case where this cost is infinite and therefore any price assignment is irrevocable. In other words, the assigned price is advertised and must be honoured as advertised even after other products appear on the market. Also, when a good appears, it comes with a survey that specifies how much various customers desire it, i.e. with a complete apecification of its utilities to all agents.

As we mentioned, in our setting the algorithm assigns prices trying to approximate the price equilibrium that we would get in an offline setting. It quickly becomes obvious that in the setting described above, the exact equilibrium are not generally computable. Hence we appeal to the concept of an *approximate* equilibrium as introduced by Deng, Papadimitriou and Safra [5] (precise definitions are given in Section 2). Prices set by online algorithms will not necessarily correspond to an exact market equilibrium, but we may still be looking for the best possible approximation, which is an approximate equilibrium. The two main parameters of an approximate equilibrium are market clearence and optimizing the pay-off for each agent (personal optimal bundle). We distinguish between two types of approximate market clearance: we can require that every good is cleared to a certain extent, or we can require that on average the goods clear. The former definition associates each good with a distinct seller and is the one adopted in the study of (off-line) equilibria, while the latter captures a setting in which there is only one seller that is satisfied when goods are cleared on average.

Our model is quite different from previous auction-type algorithms [2] for market clearence, based on online matching algorithms. Those models enforce the extra restriction that a price *and* an allocation or sale must be done online in an irrevocable way. Our model only requires that the price is set and advertised, while the allocation of goods, or the sales can be done at the end when all prices have been set. This gives a characterization of the best approximation of equilibrium prices in a different market setting, and our results indicate that it is much more difficult to get good equilibrium approximations.

Previous Work. The Arrow-Debreu theorem [1] states that in general divisible markets, equilibrium prices always exist. The theorem applies in the general setting in which each agent (buyer) has an initial endowment of goods, which she can then trade to acquire other goods, but it also applies to markets in which the agents have money with which they buy goods (and hence there is a clear distinction between sellers and buyers). Such markets are called *Fisher* markets [3, 6], and they are focus of this paper.

Computing market equilibria has been a long-standing problem in economics [9] but it was only recently that it was approached from the point of view of algorithmic solutions with strict guarantees. Deng, Papadimitriou and Safra [5] introduced the concept of an approximate equilibrium: essentially, one seeks an allocation of goods which approximately clears the market and for which every agent is approximately maximally happy. Deng *et al* considered the problem of approximating equilibria in endowment markets with linear utility functions (linear markets). They showed that, for indivisible goods, the problem is NP-hard to approximate within $1/3$, and it is NP-hard even when the number of agents is two. In contrast, for the case where the number of agents is fixed, they provided a $(1+\epsilon)$-approximation. For divisible goods they showed an exact (polynomial-time) algorithm provided that the number of goods, or the number of agents, is bounded.

For linear Fisher markets with divisible goods, Devanur, Papadimitriou, Saberi and Vazirani [4] presented a polynomial-time exact algorithm based on the primal-dual schema. Jain, Mahdian and Saberi [8] used the algorithm of [4] to provide a FPTAS in the more general setting of endowment markets. This algorithm was recently improved to a strongly polynomial algorithm by Devanur and Vazirani [6]. More work on market equilibria was presented in [7] on the spending constraint model.

Our Results. We provide upper and lower bounds on the deviation-from-equilibrium of an algorithm for the problem with respect to the two definitions of clearance. Here, the deviation-from-equilibrium is defined as the smallest k for which the prices are $(1-1/k)$-equilibrium prices (the precise definition of this measure and its rational is given in Section 2). For deterministic algorithms we show a (tight) bound of $\Theta(\min\{\sqrt{m}, n\})$ for both individual and global clearance (n and m are the number of agents and goods respectively). We then turn our attention to the application of randomization in the context of this problem; this is motivated by the observation that an algorithm with access to random bits could possibly avoid (on an average case) bad prices for the whole sequence of goods. We show a randomized algorithm that acheives a better deviation-from-equilibrium that its deterministic counterpart, when clearence is required "on the average". Specifcally, we show how to get deviation-from-equilibrium $O(\min\{m^{1/3}\sqrt{\log m}, n\})$ albeit for a somewhat restricted, but still fairly broad and natural family of inputs. Furthermore, we provide a lower bound of $\Omega(\min\{m^{1/3}, \sqrt{n}\})$. For the case of individual clearance we show that randomization does not actually help, as we show a lower bound of $\Omega(\min\{\sqrt{m}, n\})$ to the deviation-from-equilibrium under this criterion.

2 Problem Definition and Preliminaries

A market consists of a set A of n *agents* and a set G of m divisible *goods*. Each good is characterized of its *size* b_j. Agent i possesses a certain amount of *money* $e_i \in \mathbb{R}^+$, which she can use to buy goods in G. A *bundle* of goods for agent i is a vector $\mathrm{x}_i \in \mathbb{R}^m$ such that $\mathrm{x}_{ij} \leq b_j$. A *feasible allocation* (or simply allocation) $\mathbf{x}$

is a collection of n bundles $\mathrm{x}_1, \ldots, \mathrm{x}_n$ (one for each agent), such that, for every j, $\sum_{i=1}^{n} x_{ij} \leq b_j$. The *utility function* of i is a function $u_i : \mathbb{R}^m \to \mathbb{R}$; in particular, $u_i(\mathrm{x}_i)$ specifies the utility of agent i for bundle x_i (informally, it represents the happiness of i if she buys a quantity $x_{ij} \leq b_j$ of good j). Throughout this paper, we assume *linear* utility functions, that is $u_i(\mathrm{x}_i) = \sum_{j=1}^{m} u_{ij} x_{ij}$, for non-negative constants u_{ij}. We call u_{ij} the *utility of j for i*.

Suppose that good j is assigned a *price* $p_j \in \mathbb{R}^+$. Since agent i wants to maximize her utility, the *optimal bundle* for i is the bundle $\tilde{\mathrm{x}}_i$ which is the solution to the following maximization program:

$$\begin{aligned} &\text{maximize} && u_i(\mathrm{x}_i) \\ &\text{subject to} && \sum_{j=1}^{m} p_j \mathrm{x}_{ij} \leq e_i. \end{aligned} \tag{1}$$

Informally, the optimal bundle for agent i maximizes the utility i can make without taking into consideration the presence of other agents. Can we set prices so that we can find an allocation that consists of bundles that are close to optimal, for all users? Clearly, if prices are high enough then this requirement is met, as there are no conflicts between different buyers demanding more than the supply. However, we also wish to achieve *market clearance*, in the sense that there is no surplus or deficiency of goods. The concept of a *market equilibrium* aims to strike a balance between these two conflicting goals. More formally, an *ϵ-approximate equilibrium* ($0 \leq \epsilon \leq 1$), or ϵ-equilibrium for brevity, is a price vector $\mathbf{p} \in \mathbb{R}_+^m$ such that there exists an allocation $\mathrm{x} = \{\mathrm{x}_1, \ldots, \mathrm{x}_n\}$ with the following two properties:

1. For all i, $u_i(\mathrm{x}_i) \geq (1 - \epsilon) u_i(\tilde{\mathrm{x}}_i)$. $\tilde{x}_i$ is the solution to the maximization program (1).
2. The market approximately clears. Let $c_j = \frac{1}{b_j} \sum_{i=1}^{n} x_{ij}$, namely the fraction of good j that was bought. In *individual clearance* we require that $\min_j c_j \geq 1 - \epsilon$, in other words we consider the worst case clearance of the goods. Alternatively, we can look at a more relaxed definition and require that the *average* of the c_j-s is at least $1 - \epsilon$. We call this type of clearance *global clearance*.

A 0-equilibrium is simply called an *equilibrium*. We call the price vector relative to which one can find an allocation that leads to an ϵ-equilibrium an *ϵ-equilibrium price*. Without loss of generality, in linear markets and for both variants of clearance, we can normalize the size of each good to 1 unit by scaling the utilities appropriately. We notice that subject to this normalization global clearance simply states that the total number of units bought is at least $(1-\epsilon)m$. The setting of the market equilibrium problem assumes that all information is available to an algorithm for the problem. In this paper we consider the *on-line* version of the market equilibrium problem. More specifically, we assume that goods arrive on-line. Every time a good j appears, the utilities u_{ij} are revealed, for all $i \in [n]$. The on-line algorithm must assign an *irrevocable* price to each

good at the time of its appearance, that is the price of goods cannot be modified throughout the algorithm's execution[1].

We motivate our definition of a *deviation-from-equilibrium* of an on-line algorithm. When considering approximation algorithms for a maximization problem which achieves a value v while the optimal value is τ, we either say the approximation ratio is $k = \tau/v$ or, when v approaches τ, e.g. in the case of a PTAS, we rather consider that minimal ϵ for which $v \geq (1-\epsilon)\tau$. A similar situation arises here. We define the deviation-from-equilibrium of an on-line algorithm as the smallest k for which the prices set by the algorithm are $(1-1/k)$-equilibrium prices.

Additional Definitions. A good j is called *uniform* if for every two agents i, i', $u_{ij} = u_{i'j}$. A set of goods is called *monotone* if for every two goods j, j' and every two agents i, i', we have $u_{ij} \leq u_{ij'} \Rightarrow u_{i'j} \leq u_{i'j'}$. The definition asserts that the ordering of monotone goods by utility is the same for every agent. The *aspect ratio* of the market is defined as $\max_{i,j,j'} \frac{u_{ij}}{u_{ij'}}$. The good j that maximizes the ratio u_{ij}/p_j is called the *best* good for agent i.

3 Deterministic Algorithms

Theorem 1. *There exists a deterministic on-line algorithm for the problem with deviation-from-equilibrium $O(\min\{\sqrt{m}, n\})$, for individual (and therefore also for global) clearance. Furthermore, this bound is tight.*

Proof. **Upper Bound**. We consider individual clearance, which clearly implies global clearance with the same guarantees. First, we provide some intuition behind the assignment of prices. If the price of each good is set as high as E (recall that E is the total money of all agents), then clearly there is an allocation in which every agent gets as much utility as from its optimal bundle; however at most one unit of good is allocated, and we are far from market clearance. On the other extreme, if the prices are very low, e.g., E/m, then all goods can be allocated, however there will be contention between agents for goods that are important to a large subset of agents. We reconcile the two extremes by assigning prices as follows: the price of the j-th good, for all $j \leq n^2$ is set to $\frac{E}{\sqrt{j}}$. For more details see the full paper.

Lower Bound. The adversary will present to n agents, each having a unit amount of money, a sequence of m uniform goods; that is, for every good j and agent i, $u_{ij} = u_j$. The intuition behind the adversarial input is that when the algorithm considers a good of very high utility *for all* agents, then it has to set a high price to it, otherwise there will be heavy contention between agents. By

[1] Alternatively, one could assume a model in which agents appear on-line, each revealing how much utility she can make from each good. Such a model would make sense only if the price of every good is set before any allocation (even for the very first agent) takes place. However, if prices are set, the problem is trivial to solve for linear markets, by using linear programming.

providing a sequence of "progressively better" goods, in terms of their utility, the adversary will force the algorithm to assign high prices to every good, which implies poor market clearance. For more details see the full paper.

The upper bound of Theorem 1 is tight, provided that the aspect ratio of the market is sufficiently high. What if the aspect ratio is bounded? The following theorem builds upon the idea behind the lower-bound proof of Theorem 1. The intuition here is that goods in the sequence become progressively better, but only as much as possible given the bound on α.

Theorem 2. *The deviation-from-equilibrium of every deterministic algorithm for a market with aspect ratio $\alpha > 1$ is in $\Omega(\min\{nb, \sqrt{mb}\})$, where $b = \frac{\beta^m(\beta-1)}{\beta^{m+1}-1}$, and $\beta = \alpha^{\frac{1}{m}}$.*

Proof. We will consider global clearance; the lower bound then carries over to individual clearance. As in the proof of Theorem 1 the adversary will present to n agents, each having a unit amount of money, a sequence of m uniform goods (we will denote by u_j the utility of the j-th good in the sequence, with $u_1 = 1$). Every time the algorithm assigns a "low" price to good $j < m$, then good $j+1$ is is such that $u_{j+1} = \beta \cdot u_j$ (hence if all j first goods were assigned low prices, then $u_{j+1} = \beta^j$). Otherwise, the adversary presents $m - j$ goods, each of (low) utility β^{j-m}, and terminates the game. We will assume (without loss of generality) that the algorithm knows m in advance, and that the deviation-from-equilibrium of the algorithm, say k, for $k \geq 1$ is a function of m, n, $\mathbf{e}$ and α *only*.

Consider good j in the sequence, assuming that no good in $[j]$, with $l \leq m-1$ has received a low price. We claim that if the price p_j is "low" this provides a lower bound to the deviation-from-equilibrium k by the following relaton.

$$\beta^j \cdot \min\{1, \frac{1}{p_j}\} < k \cdot (\frac{\sum_{l=1}^{j} \beta^l + (m-j)\beta^{j-m}}{n}). \tag{2}$$

To see this, note that in such a case, the adversary will terminate the game by providing $m - j$ goods of utility β^{j-m}. Then, for every allocation of the m goods to agents, there exists one agent, say i, who will receive utility at most $(\sum_{l=1}^{j} \beta^l + (m-j)\beta^{j-m})/n$. On the other hand, the optimal bundle for i yields utility at least equal to the LHS of (2). Taking into account the fact that $\frac{\beta^j}{\sum_{l=1}^{j} \beta^l + (m-j)\beta^{j-m}} \geq \frac{\beta^m}{\sum_{l=1}^{m} \beta^l} = b$, we conclude that, for the algorithm to have deviation-from-equilibrium k, the price p_j must not be low, namely it must be such that

$$n \min\{1, \frac{1}{p_j}\} \cdot b \leq k, \tag{3}$$

If $p_j \leq 1$, (3) gives $k \geq nb$. Otherwise, (3) implies that $p_j \geq \frac{n}{k} b$, for all $j \in [m-1]$. Hence at most $(Ek)/(bn) + 1 = k/b + 1$ goods can be allocated to agents, and since we require that the algorithm has deviation-from-equilibrium k, it must be that $\frac{m}{k/b+1} \leq k$, thus $k \in \Omega(\sqrt{mb})$. Summarizing, $k \in \Omega(\min\{nb, \sqrt{mb}\})$.

Theorem 2 demonstrates that when α is exponential on m, then the algorithm used in the proof of Theorem 1 is asymptotically optimal.

It is worth mentioning that for the special case where all goods are identical (and hence the only information not known to the on-line algorithm is the number of goods), a variant of this approach can be employed to show that no deterministic algorithm has deviation-from-equilibrium better than a certain constant bigger than 1. We omit the details.

4 Randomized Algorithms

A randomized on-line algorithm is an algorithm which assigns irrevocable prices to goods according to a certain probability distribution. We start by stating precisely the definition of deviation-from-equilibrium of such an algorithm. Consider an allocation of goods after the prices have been set. Let F_i denote the random variable which is the utility agent i can make from this allocation. Let also G_i denote the random variable that corresponds to the utility of the optimal bundle for agent i. Last, define H_j to be the random variable which denotes how much of good j was bought, and $H = \sum_j H_j$. Then, $K_G = \max\{\max_i G_i/F_i, m/H\}$ and $K_I = \max\{\max_i G_i/F_i, \max_j 1/H_j\}$ are the random variables that denote the global and individual clearance respectively. Let $k_G = \mathbb{E}[K_G]$ and $k_I = \mathbb{E}[K_I]$ be the corresponding expectations of these variables and they are the ones we consider. We start with a negative result.

Theorem 3. *The deviation-from-equilibrium of every randomized on-line algorithm for markets with global clearance $\Omega(\min\{m^{\frac{1}{3}}\sqrt{n}\})$. For markets with individual clearance it is $\Omega(\min\{\sqrt{m}, n\})$.*

Proof. Using Yao's principle, we present a distribution on inputs on which every deterministic algorithm has high expected deviation-from-equilibrium . Let I_j be the input $u_1 << u_2 << \ldots << u_j$ and $u_{j+1}, \ldots, u_m = 0$. Take input I_j for $j < m$ with probability $\frac{1}{2m}$ and I_m with probability $\frac{1}{2} + \frac{1}{2m}$. Let $p_1, p_2, \ldots, p_m$ be the algorithm answers to I_m. There is a subtle point to note here: since I_j and I_m are consistent for the first j goods, it must be the case that prices set by the algorithm for the input I_j are with agreement to those set for the input I_m, namely $p_1, \ldots, p_j$. Arguments similar to the ones in 2 show that considering agent j, at the event of input I_j we have the bound $K_G, K_I \geq n \cdot \min\{1, 1/p_j\}$, and at the event of inout I_m we have (from clearance constriants) that $K_G \geq m/l$ where l is the maximal number of goods totaling to at most n. So

$$k_G = \mathbb{E}[K_G] \geq \max\left\{\frac{n}{2m}\sum_i \min\{1, 1/p_j\}, \frac{m}{2l}\right\}.$$

We apply the first part of Lemma 1 to conclude $k_G \geq \frac{1}{2}\min\{\sqrt{n}, m^{\frac{1}{3}}\}$.

Considering K_I, we notice that if the individual clearance is at most K, a quantity of $1/K$ of each good must be purchased by the agents. This means that $\frac{1}{K} \cdot \sum_i p_i \leq E = n$, and therefore we get

$$k_I = \mathbb{E}[K_I] \geq \max\left\{\frac{n}{2m}\sum_i \min\{1, 1/p_j\}, \frac{\sum_i p_i}{2n}\right\}.$$

By the second part of Lemma 1 we get $k_I = \Omega(\min\{\sqrt{m}, n\})$.

Lemma 1. *Let n, m be positive integer numbers, p_i be nonnegative reals, and $l = l(p_1, \ldots, p_m; n)$ is the maximal l such that the sum of the l smallest p_i does not exceed n. Then*

$$\max\left\{\frac{n}{m}\cdot\sum_i \min\{1, 1/p_i\}, \frac{m}{l}\right\} \geq \min\{\sqrt{n}, m^{\frac{1}{3}}\},$$

and also

$$\max\left\{\frac{n}{2m}\sum_i \min\{1, 1/p_j\}, \frac{\sum_i p_i}{2n}\right\} = \Omega\{m^{1/2}, n\}.$$

Proof. Let $S = \{j : p_j < 1\}$ and L be the set if indices of the l smallest p_i-s (cutting ties arbitrarily). Further, let $B = L \setminus S$, $s = |S|$ and $b = |b|$. Clearly $l = s + b$. Now,

$$\sum_{i\in B} \min\{1, 1/p_i\} = \sum_{i\in B} 1/p_i = b \cdot \frac{\sum_{i\in B} 1/p_i}{b} \geq b \cdot \frac{b}{\sum_{i\in B} p_i} \geq b^2/n.$$

The first inequality above is the Arithmetic-Harmonic-Mean inequality. So

$$\sum_i \min\{1, 1/p_i\} \geq \sum_{i\in S} \min\{1, 1/p_i\} + \sum_{i\in B} \min\{1, 1/p_i\} \geq s + b^2/n$$

and we get

$$\max\left\{\frac{n}{m}\cdot\sum_i \min\{1, 1/p_i\}, \frac{m}{l}\right\} \geq \max\left\{\frac{n}{m}\cdot(s + b^2/n), \frac{m}{s+b}\right\}.$$

It is now enough to show that for every choice of nonnegative b and s the inequality $\max\left\{\frac{n}{m}\cdot(s + b^2/n), \frac{m}{s+b}\right\} \geq \min\{m^{\frac{1}{3}}, \sqrt{n}\}$ holds. First assume $s = 0$. Here we need to optimize $\max\{b^2/m, m/b\}$ which is clearly at least $m^{\frac{1}{3}}$. We now turn to the case $s > 0$. We will use the simple inequality $\max\left\{\frac{n}{m}\cdot(s + b^2/n), \frac{m}{s+b}\right\} \geq \frac{1}{2}\left(\frac{n}{m}\cdot(s + b^2/n) + \frac{m}{s+b}\right)$, and will now lower bound the latter expression. Assume b is fixed and we need to find that value of s minimizing the expression. Easy calculus shows that $s = m/\sqrt{n} - b$ is that value. We can safely assume $m/\sqrt{n} - b > 0$ otherwise $s = 0$ would be the best choice which is a case we already covered. Substituting for s we get

$$\frac{n}{m}\cdot(s + b^2/n) + \frac{m}{s+b} \geq \frac{n}{m}\cdot(m/\sqrt{n} - b + b^2/n) + \sqrt{n}.$$

Now we optimize over b and get that $b = n/2$ is the minimizing value for the last expression. By the same argument we may assume here that $n/2 \leq m/\sqrt{n}$. We substitute b for $n/2$ and get

$$\frac{n}{m} \cdot (m/\sqrt{n} - b + b^2/n) + \sqrt{n} = 2\sqrt{n} - \frac{1}{4}n^2/m \geq \sqrt{n}.$$

For the second part of the lemma, Using the same definition $S = \{j : p_j < 1\}$ we get

$$\sum_i \min\{1, 1/p_i\} = \sum_{i \in S} \min\{1, 1/p_i\} + \sum_{i \notin S} \min\{1, 1/p_i\} \geq s + \sum_{i \notin S} 1/p_i \geq$$

$$\geq s + \frac{(m-s)^2}{\sum_{i \notin S} p_i} = s + \frac{(m-s)^2}{P},$$

where $s = |S|$ and $P = \sum_{i \notin S} p_i$. Now

$$k \geq \frac{1}{2} \max \left\{ \frac{n}{m} \sum_i \min\{1, 1/p_j\}, \frac{\sum_i p_i}{n} \right\} \geq \frac{1}{4} \left(\frac{n}{m} \sum_i \min\{1, 1/p_j\} + \frac{\sum_i p_i}{n} \right)$$

$$\geq \frac{1}{4} \left(\frac{n}{m}(s + (m-s)^2/P) + \frac{P}{n} \right) = \Omega\{m^{1/2}, n\}.$$

The last quantity is easily verified by checking separately for $s < m/2$ and for $s \geq m/2$.

Can we get a better upper bound by using randomization? The following result shows that for monotone goods the answer is positive. It should be noted that the adversarial input of both our deterministic and randomized lower bounds complies to the condition of monotonicity.

Theorem 4. *For markets with* monotone *goods and* global *clearance, there exists a randomized on-line algorithm with deviation-from-equilibrium* $O(\min\{m^{1/3}\sqrt{\log m}, n\})$.

Proof. We first show how to get a $O(\min\{m^{1/3} \log m, n\})$ first. Let $\tilde{j}$ denote $2^{\lceil \log j \rceil}$. The algorithm assigns prices to goods according to the following probability distribution. For every good $j \leq n^3$

$$p_j = \begin{cases} E/\tilde{j}^{2/3} \text{ with probability } 1/\tilde{j}^{1/3} \\ E/\tilde{j}^{1/3} \text{ otherwise} \end{cases}$$

We call a good *cheap* when it receives price $E/\tilde{j}^{2/3}$ and *expensive* otherwise[2]. Next, we set the prices of all goods $j > n^3$, if any, to be arbitrarily low; we call such goods *free* goods.

[2] Note that by this definition there are cheap goods that cost more than expensive ones.

First, we want to show that with this setting the market approximately clears. Suppose first that $m \leq n^3$. Consider the set of goods $B = \{j : \tilde{j} \geq m/4\}$ and let B_c be the cheap goods in B. Clearly $|B| \geq m/2$ and that $\mathbb{E}[|B_c|] \geq |B|/(2m)^{\frac{1}{3}} = \Omega(m^{2/3})$. Further, Chernoff bound guarantees that $\mathbb{P}[|B_c| \leq \frac{1}{2}\mathbb{E}[|B_c|] = \exp(-\Omega(m^{2/3}))$. Look at the following strategy. It is easy to see that agents can allocate $1/4$ of their money to buy $\Omega(m^{1/3})$ goods j for which $m/8 \leq \tilde{j} < m/4$, as the prices for these goods is at most $O(E/m^{1/3})$. Another $1/4$ is used to buy as many goods in B_c. Since prices there are $O(E/m^{2/3})$ we can buy as many as $\min\{|B_c|, m^{2/3}\}$ such goods. We get that the number of goods that can be bought g is $\Omega(m^{2/3})$ with probability $1 - \exp(-\Omega(m^{2/3}))$ and $\Omega(m^{1/3})$ otherwise which gives $\mathbb{E}[m/g] = \Omega(m^{1/3})$. In the case where $m > n^3$ the agents can still achieve an expected ratio of $n^{3 \cdot \frac{1}{3}} = n$ on the first n^3 by exactly the same argument, and then buy all $m - n^3$ free goods at arbitrary low cost which can only improve the ratio. Hence, we only need to be concerned about finding an allocation that yields high utility to each agent. Since each agent spent at most half of her money for the allocations above and since constant factors do not affect our bounds, we may assume for simplicity that these allocations did not cost money at all.

As was shown in Theorem 1, regardless of the prices a factor n of the performance can always be guaranteed. We now get to the interesting part of the theorem where $m \leq n^3$. Notice that we can think of the goods as organized in *bunches* by their $\tilde{j}$ value. Namely the bunches are of size $1, 2, 4, \ldots$ (with the exception of the last one which may be smaller). In the allocation we will describe, every agent will spend at most a fraction of $1/w$ of her money, for some $w > 1$ to a good that has price E/w. Whenever all agents comply to this restriction there is no deficiency of the goods, or informally, there is no contention between agents about any good, and we can consider the allocation of each agent individually.

Let j_i be any of the best goods for agent i; recall that these are the good maximizing the ratio utility per price. In the event where j_i is an expensive good, its price is at least $E/m^{1/3}$. In this case agent i will spend $e_i/m^{1/3}$ on buying j_i and so she receives at least $1/m^{1/3}$ of her optimal utility. Otherwise j_i is a cheap good. We now introduce some additional notation. Let u_i be the utility of j_i, b_i the bunch containing j_i and C_i are the goods in bunch b_i with utility at least u_i. Finally, let $t_i = \min\{|C_i|, T_i\}$, S_i be the size of bunch b_i and $T_i = S_i^{1/3}$. Agent i spends e_i/T_i on each of t_i goods from C_i. Therefore the quantity of goods in C_i which agent i gets is exactly $t_i \cdot \frac{e_i/T_i}{E/T_i} = e_i t_i/E$, and since the goods in C_i have at least utility u_i we get a total utility of at least $u_i e_i t_i/E$. On the other hand, an upper bound to the optimal utility for agent i is $\frac{u_i e_i}{E/T_i^2}$ Letting K_i be the ratio between the utility from optimal bundle and utility from allocation for i we get

$$K_i \leq \frac{T_i^2 e_i u_i}{e_i u_i t_i} = T_i^2/t_i. \tag{4}$$

At this point, we show that instead of considering the n different random variables K_i we can look at $\log m$ very related variables: For each bunch b let us choose an ordering that is consistent with the utilities for all agents. In other

words if j and j' are goods in bunch b and $u_{ij} > u_{ij'}$ then good j appears before j' in the ordering. Such ordering is possible thanks to the monotonicity of goods. Let r_b be the minimal ordinal of a cheap good according to the ordering. As usual S_b is the size bunch b and $T_b = S_b^{1/3}$. Finally, let $\nu_b = \min\{r_b, T_b\}$. It is now easy to see that if $b_i = b$ then $\nu_b \leq t_i$. Defining $L_b = \max\{K_i : b_i = b\}$ we now get that

$$L_b \leq \max_{\{i|b_i=b\}} T_i^2/t_i \leq T_b^2/\nu_b.$$

The maximum ratio of optimal utility over utility through allocation over all agents is now

$$\max\{m^{1/3}, \max_i K_i\} = \max\{m^{1/3}, \max_b L_b\} \leq m^{1/3} + \sum_b L_b.$$

and so $\mathbb{E}[K] \leq m^{1/3} + \sum_b \mathbb{E}[T_b^2/\nu_b]$.

Lemma 2. *Let $S = \{1, 2, \ldots, s\}$ and pick elements in S independently with probability $q = 1/T$ where $T \leq s$. Let r be the minimal number picked or s if none exists. Also, let $\nu = \min\{r, T\}$. Then $\mathbb{E}[1/\nu] = \log T/T$.*

Proof. As long as $r \leq T$, it is distributed geometrically with parameter q. Now

$$\mathbb{E}[1/\nu] = \mathbb{P}[r \leq T] \cdot \mathbb{E}[1/\nu | r \leq T] + \mathbb{P}[r > T] \cdot \mathbb{E}[1/\nu | r > T] \leq$$
$$\leq \sum_{r=1}^{T} \frac{q(1-q)^r}{r} + 1/T \leq qH(T) + 1/T \leq O(\log T/T).$$

Applying the lemma we get that $\mathbb{E}[1/\nu_b] = O(\log T_b/T_b)$. Summing up we have

$$\mathbb{E}[K] \leq m^{1/3} + \sum_b \mathbb{E}[T_b^2/\nu_b] \leq m^{1/3} + \sum_b O(T_b \log T_b) = O(m^{1/3} \log m).$$

To get the desired $O(m^{1/3}\sqrt{\log m})$ bound we notice the imbalance between the two criteria for the equilibrium. We can multiply the prices of cheap goods by a factor of $\sqrt{\log \tilde{j}}$. Obviously the clearance does not deteriorates by more than a factor of $\sqrt{\log m}$. On the other hand, we get the improved upper bound for L_b for the second criterion, $L_b \leq T_b^2/(\sqrt{\log S_b}\nu_b)$. From this the improvement in the bound follows easily.

5 Discussion and Future Work

In this work we formulate and address the problem of on-line equilibria in linear Fisher markets with divisible goods. A number of unresolved issues remain; these are not only of theoretical interest, but they relate to situations which are expected to occur in practice. Among them we first distinguish the following two: First, is it possible to show better (deterministic or randomized) upper bounds for markets with subexponential aspect ratio α? Ideally, we would like to provide an on-line algorithm, which has no knowledge of the aspect ratio of the entire sequence, and whose deviation-from-equilibrium is a function of α.

Second, is it possible to remove the monotonicity condition from Theorem 4? The result is meant to show that randomization is helpful when considering global clearance, but we would like to extend it to capture more general markets. We believe that an elaborate probabilistic argument will be needed to address this issue.

A different extension deals with markets with indivisible goods. We have some preliminary results for this type of markets. It is worth noting that here one has to provide certain restrictions on the sequence of the goods the adversary will provide, otherwise no on-line algorithm may achieve bounded deviation-from-equilibrium . Note also that since an (exact) equilibrium does not necessarily exist in this setting, we must relate to the best possible (approximate) equilibrium that an off-line algorithm can achieve.

As argued in the introduction, our work is motivated by dynamic markets, where a cost is associated with any change in the price of an existing good. Our on-line upper and lower bounds address the case in which the cost is infinite. But what about other cost functions? For instance, suppose that the cost to change a price is constant, and there is a strict bound on the total cost due to price changes every time a good arrives. What is the best approximate equilibrium we can guarantee in such a setting?

References

1. K. Arrow and G. Debreu. Existence of an equilibrium for a competitive economy. *Econometrica*, 22:265–290,1954.
2. Avrim Blum and Tuomas Sandholm and Martin Zinkevich, Online algorithms for market clearing, *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, 2002, pp 971–980
3. W.C. Brainard and H.E. Scarf. How to compute equilibrium prices in 1891. *Cowles Foundation Discussion Paper 1270*, 2000.
4. N. Devanur, C. Papadimitriou, A. Saberi and V. Vazirani. Market equilibrium via a primal-dual type algorithm. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 389–395, 2002.
5. X. Deng, C. Papadimitriou and S. Safra. On the complexity of equilibria. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pp. 67–71, 2002.
6. N. Devanur and V. Vazirani. An improved approximation scheme for computing market equilibrium. In *Proceedings of the 23rd Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2003.
7. Nikhil R. Devanur, Vijay V. Vazirani. Extensions of the spending constraint-model: existence and uniqueness of equilibria (extended abstract). *ACM Conference on Electronic Commerce* 2003: 202-203
8. K. Jain and M. Mahdian and A. Saberi. Approximating market equilibria. In *Proceedings of the 6th Workshop on Approximation algorithms for Combinatorial Optimization Problems (APPROX)*, 2003.
9. H.E. Scarf. *The computation of Economic Equilibria (with collaboration of T. Hansen)*. Cowles Foundation Monograph No. 24. Yale University Press, 1973.
10. L. Walras. *Éléments d'économie politique pure; ou, théorie de la richesse sociale (Elements of Pure Economics, or the theory of social wealth)*. Lausanne, Paris, 1874. (1899, 4th ed.; 1926 rev. ed. , 1954 Engl. transl.).

Interval Subset Sum and Uniform-Price Auction Clearing*

Anshul Kothari[1], Subhash Suri[1], and Yunhong Zhou[2]

[1] Computer Science Depart, University of California, Santa Barbara, CA 93106
{kothari,sur}@cs.ucsb.edu
[2] HP Labs, 1501 Page Mill Rd, Palo Alto, CA 94304
yunhong.zhou@hp.com

Abstract. We study the *interval subset sum problem* (ISSP), a generalization of the classic *subset-sum* problem, where given a set of intervals, the goal is to choose a set of integers, *at most one from each interval*, whose sum best approximates a target integer T. For the *cardinality constrained* interval subset-sum problem (kISSP), at least $k_{\min}$ and at most $k_{\max}$ integers must be selected. Our main result is a fully polynomial time approximation scheme for ISSP, with time and space both $O(n \cdot 1/\varepsilon)$. For kISSP, we present a 2-approximation with time $O(n)$, and a FPTAS with time $O(n \cdot k_{\max} \cdot 1/\varepsilon)$.

Our work is motivated by auction clearing for uniform-price multi-unit auctions, which are increasingly used by security firms to allocate IPO shares, by governments to sell treasury bills, and by corporations to procure a large quantity of goods. These auctions use the *uniform price* rule – the bids are used to determine who wins, but all winning bidders receive the same price. For procurement auctions, a firm may even limit the number of winning suppliers to the range $[k_{\min}, k_{\max}]$. We reduce the auction clearing problem to ISSP, and use approximation schemes for ISSP to solve the original problem. The cardinality constrained auction clearing problem is reduced to kISSPand solved accordingly.

1 Introduction

We introduce the *interval subset sum problem* (ISSP), where given a set of intervals $[a_i, b_i]$, for nonnegative integers a_i, b_i, $i = 1, 2, \ldots, n$, and a target integer T, the goal is to choose a set of integers, at most one from each interval, that best approximates T. ISSP is very similar to the classical *subset sum problem*, with each integer item replaced by an interval of integer values. By limiting $a_i = b_i$ for all i, ISSP degenerates into the subset sum problem. Because the subset sum problem is NP-complete [7], it is easy to show that ISSP is also NP-complete. In this paper we study the optimization version of ISSP, and present approximation schemes for it.

* Kothari and Suri were partially supported by NSF grants CCR-9901958 and IIS-0121562. Most of the work was done while Kothari was an intern at HP Labs.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 608–620, 2005.

ISSP is motivated by uniform-price multi-unit combinatorial auctions. In many practical scenarios, additional requirements are imposed on the structure of the optimal solution. A quite frequent and natural restriction concerns the number of items included in an optimal solution. Therefore, we also consider the *cardinality constrained* version of ISSP, where a feasible solution should contain at most $k_{\max}$ integers and at least $k_{\min}$ integers. We call this the *cardinality constrained interval subset sum problem* (kISSP) and will also present approximation algorithms for it.

1.1 Motivations

The problem we consider in this paper is motivated by single-item multi-unit auctions, where the auctioneer wants to allocate multiple indistinguishable units among a set of bidders, optimizing certain metrics. The metric can be revenue maximization as in the case of forward auctions like IPO auctions, treasury auctions etc or cost minimization as in the case of reverse auctions like procurement auctions.

One of the most important aspects of any auction is bidding format or how does a bidder express its bid. Traditionally, single item auctions have used *point bid* format, where a bid is a pair (p, u), where p is the per unit price and u is the number of units the bidder is interested in. The point bidding language is very restrictive as the auctioneer can allocate exactly u units to the bidder or nothing at all. In practice, a bidder might be satisfied with some thing close to u. A more expressive bidding format is *interval bids*, where the bidder specifies a range $[u_1, u_2]$ such that he can be allocated any number of units in this range. This type of bidding language was implicitly used by Google for its IPO auction. In its prospectus [17] Google mentioned that a winning bid with asking quantity u will be guaranteed to receive at least 80% of that number, i.e., any quantity in the interval $[0.8u, u]$.

Another important application of interval bids is procurement auctions used by firms to procure raw materials. In these auctions, the *suppliers* wish to express *volume discounts* in their bids: instead of offering a fixed price per unit, they prefer a more expressive bidding language. The language most widely used has the form of a *piecewise constant* curve. (See Figure 1.) Such a bid can be written as a list of tuples $\langle (u_1, p_1), (u_2, p_2), \ldots, (u_{m-1}, p_{m-1}), (u_m, \infty) \rangle$, where $u_1 < u_2 < \ldots < u_m$ and $p_1 > p_2 > \ldots > p_{m-1}$. The procurement interpretation is that price per unit is p_i for any number of units in the range $[u_i, u_{i+1})$, and u_m is the maximum number of units offered by this bid. As the quantity increases, the unit price decreases (volume discount).

Procurement auctions usually have one additional side constraint. The buyer wishes to control the number of different suppliers that win in the auction: managing many different suppliers is costly, but too few suppliers expose the firm to vulnerability in case of unforeseen events. Thus, the firm puts a constraint that the number of *winning suppliers* be in the range $[k_{\min},\ k_{\max}]$, where $k_{\min}, k_{\max} > 0$ are integers.

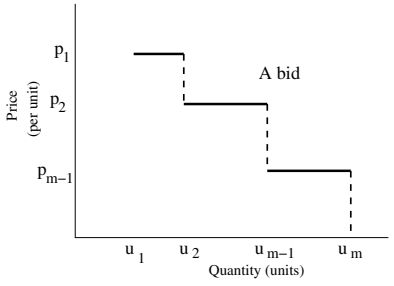


Fig. 1. A piecewise constant bid.

Another important aspect of a single-item multi-unit auction is clearing price. There are two kinds of pricing rules: *uniform* and *non-uniform*. In this paper, we consider the *uniform-price* rule: all winners receive the same per unit price. The bids are used to determine who all win the auction and how many units are allocated to them but all the winners receive a common per unit price. This pricing rule has the advantage of being *ex ante* fair, and promotes goodwill among auction participants. Auction sites like *MobShop* use such pricing rules for group buying. For the non-uniform pricing rule, where different winners are offered different prices, the auction problem translates into a generalized form of the knapsack problem, which we have studied in a companion paper [11].

1.2 Related Work

The subset-sum problem is a special case of the 0-1 Knapsack problem where item profits and item weights coincide. A large number of theoretical and practical papers have appeared on variants of knapsack problems. See Kellerer, Pferschy and Pisinger [9] for an extensive treatment of knapsack problems. The subset-sum problem is well-known to be NP-complete [7]. The classical dynamic programming (DP) approach gives a pseudo-polynomial time algorithm with running time $O(nT)$. An optimal algorithm with improved complexity is due to Pisinger [15]. The first fully polynomial time approximation scheme for the subset-sum problem was suggested by Ibarra and Kim [6]. After a series of improvement, the best currently known result is by Kellerer et al [8]. The algorithm finds an approximate solution with relative error less than ε in time $O(\min\{n \cdot 1/\varepsilon, n + 1/\varepsilon^2 \log(1/\varepsilon)\})$ and space $O(n + 1/\varepsilon)$.

Cardinality constraints arise from column generations of cutting stock problems, and were originally studied by Gilmore and Gomory [5]. For the knapsack problem with cardinality constraints, it is formally introduced by Caprara et al. [1], where an upper bound $k_{\max}$ is given on the number of packed items. Caprara et al. also study the cardinality constrained subset-sum problem, and it runs in $O(n \cdot 1/\varepsilon \cdot \ell)$ time and $O(n + \ell^2/\varepsilon)$ space, where $\ell = \min\{1/\varepsilon, k_{\max}\}$. Our work generalizes items into intervals, and considers both an upper bound ($k_{\max}$) and a lower bound ($k_{\min}$) on the number of items selected, thus it demands extra techniques to handle.

There has been a flurry of research activities for large scale procurement in recent years, inspired by the emergence of electronic commerce. In particular, *combinatorial auctions* have been proposed as expressive, economically efficient, and truthful mechanisms for resource distributions [10, 12, 14, 16]. The winner determination problem in combinatorial auctions, unfortunately, is NP-complete and inapproximable [13] in general. Consequently, there is enormous interest in finding the right level of generality at which to address the problem. The setting studied in our paper is eminently practical; for various reasons, firms tend to conduct separate auctions for different goods, thus eliminating the main source of complexity in combinatorial auctions. The other two sources (bounding the numbers of winners, and the expressiveness of bidding language) remain, and our work incorporates them both.

In procurement auctions, the work most similar to ours is [2–4]. The focus and contributions of these papers, however, are very different from ours. For instance, the procurement problem studied by Eso et al. [4] is similar to ours, but they use a different volume discount model: the supplier asks for unit price p_1 up to some quantity q_1; the discount is offered *only for the additional* units beyond q_1. We believe our model is more natural and commonly used. But the main difference is that this paper contains no *algorithmic* results: *it simply formulates the problem as a general mixed integer linear program, and give some empirical results on synthetic data.* Davenport et al. [3] addresses double auctions, where multiple buyers and sellers trade a *divisible good*, and its focus is also different: it investigates the *equilibrium* prices using the demand and supply curves, whereas our focus is on cost minimization for the buyer. Dailianas et al. [2] addresses double auctions which has a more general discount model than ours but uses *heuristics* to solve the optimization problem. These heuristics have no provable theoretical performance guarantees.

The rest of the paper is organized as follows. We define ISSP and kISSP formally and obtain a canonical solution structure for them in Section 2. We present a FPTAS for ISSP in Section 3 and a FPTAS for kISSP in Section 4. We transform the uniform-price multi-unit auction clearing problem into ISSP (or kISSP) and use FPTAS for ISSP (or kISSP) to solve the original problem in Section 5, and conclude in Section 6.

2 Definitions and Preliminaries

The decision problem of ISSP is the following: given a set of intervals $[a_i, b_i]$ where a_i, b_i are positive integers for $i = 1, \ldots, n$, and a target integer T, decide whether there exists a subset of integers, at most one from each interval, such that their total sum equals to exactly T. ISSP is NP-complete as it contains the classical subset sum problem as a special case ($a_i = b_i$ for all i), which is one of the earliest examples proved to be NP-complete. So we focus on the optimization version of ISSP, and it has two variations: maximization and minimization. The maximization problem is to choose a subset S' whose elements sum up to *at most* T while *maximizing* the sum of the subset. The minimization problem is

to choose a subset S' whose elements sum up to *at least* T while *minimizing* the subset sum. Formally these optimization versions are defined as following:

Instance: A set of intervals $[a_i, b_i]$, for integers $b_i \geq a_i > 0$, $i = 1, 2, \ldots, n$, and a target integer T.
ISSP (max): Determine a set of integers $x_1, x_2, \ldots, x_n$ with $\sum_{i=1}^{n} x_i \leq T$ such that $\sum_{i=1}^{n} x_i$ is maximized, where $x_i > 0$ implies that $x_i \in [a_i, b_i]$.
ISSP (min): Determine a set of integers $x_1, x_2, \ldots, x_n$ with $\sum_{i=1}^{n} x_i \geq T$ such that $\sum_{i=1}^{n} x_i$ is minimized, where $x_i > 0$ implies that $x_i \in [a_i, b_i]$.

For the cardinality constrained interval subset sum problem (kISSP), there are also two optimization versions: maximization and minimization. The definitions are identical to the definitions for the optimization models of ISSP, with an extra cardinality constraint:

$$k_{\min} \leq |\{x_i \mid x_i > 0\}| \leq k_{\max},$$

where $|X|$ denotes the cardinality of set X.

Intuitively, the minimization version of ISSP (kISSP) corresponds to cost minimization for reverse auctions, while the maximization version corresponds to revenue maximization of forward auctions. The minimization version of ISSP (kISSP) is highly similar to its corresponding maximization version in both computational complexity and algorithmic details, thus we will focus only on the maximization version during this paper. From now on, by default ISSP (kISSP) denotes the maximization version of the corresponding optimization problem.

Our main results are fully polynomial time approximation schemes for both ISSP and kISSP. The algorithm for ISSP produces a solution that is within $(1 + \varepsilon)$ of the optimal in $O(n \cdot 1/\varepsilon)$ time[1]. The cardinality constraints add a multiplicative factor of $k_{\max}$ to the complexity, and kISSP can be solved in $O(n \cdot k_{\max} \cdot 1/\varepsilon)$ time. Thus the time complexity of these algorithms for ISSP and kISSP *closely matches* the time complexity for the best known algorithms for their corresponding classical versions of the subset sum problem.

We begin our discussion by describing an important structural property of the optimal solution, which is used by our algorithms.

2.1 A Canonical Optimal Solution

Consider a solution of ISSP (kISSP) $x_1, x_2, \ldots, x_n$. We call a non-zero element x_i an *anchored element* if $x_i = a_i$ or $x_i = b_i$; that is, the element is the leftmost or the rightmost point of its interval. Naturally, if $x_i = a_i$, we call it a *left anchor* and if $x_i = b_i$, we call it a *right anchor*. A non-zero element x_i that is not anchored will be called *midrange*. We identify intervals with the same label

[1] For a maximization problem, let A be output value of the algorithm and O be the optimum value, a $(1 + \varepsilon)$-approximation guarantees that $O \leq A(1 + \varepsilon)$, i.e., $A \geq O/(1 + \varepsilon)$. In this paper, if $A \geq (1 - \varepsilon)O$, we also consider it as a $(1 + \varepsilon)$-approximation. These two definitions are asymptotically equivalent.

as the label of the corresponding element; that is, if x_i is left-anchored, then we say that $[a_i, b_i]$ is left-anchored etc. The following fact is straightforward.

Fact 1 *There exists an optimal solution of ISSP (kISSP) with at most one midrange element.*

If an optimal solution contains only anchored elements, we can pick one of the anchored elements as a proxy midrange element. Therefore, for the rest of the discussion, we assume that the optimal solution has exactly a midrange element. Our next lemma characterizes a property that will be a key to our algorithm. It states that if we consider the input intervals in some arbitrary order, then there exists an optimal solution such that the midrange element separates the right-anchored intervals from the left-anchored intervals. Figure 2 shows an example with 5 intervals, and target $T = 385$. A canonical solution is shown where $[50, 65]$ is the midrange interval, while $[30, 38]$ is left-anchored, and $[200, 300]$ is right-anchored.

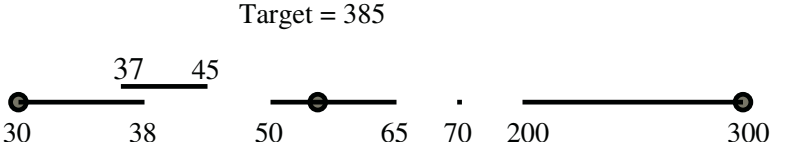


Fig. 2. A canonical optimal solution. Canonical solutions are not unique.

Lemma 1 (Canonical Solution). *Consider the intervals* $\{[a_i, b_i] \mid 1 \leq i \leq n\}$ *in some arbitrary order. There exists an optimal solution of ISSP (kISSP) with the following property: if* x_r *is the midrange element in the solution, then all the left-anchored intervals precede* $[a_r, b_r]$*, and all the right-anchored intervals follow* $[a_r, b_r]$*.*

3 Approximation Schemes for ISSP

In this sections we consider approximation algorithms for ISSP. We first give a relatively simple algorithm, which gives a fully polynomial time approximation scheme with running time $O(n \cdot 1/\varepsilon^2)$. Then we use more involved analysis to give another fully polynomial time approximation scheme with running time $O(n \cdot 1/\varepsilon)$.

3.1 A Simpler FPTAS

We begin with some simple observations. If $a_i \leq T \leq b_i$ then there is a trivial solution using just this interval. If $a_i > T$, then x_i must be 0 and we can ignore the interval $[a_i, b_i]$. Therefore, we assume that for all intervals, $b_i < T$. Furthermore, the problem is non-trivial only if $T < \sum_{i=1}^{n} b_i$. This follows because if $T \geq \sum_{i=1}^{n} b_i$, then the optimal solution is to choose $x_i = b_i$, for $i = 1, \ldots, n$. If

$\sum_{i=1}^{n} a_i \le T \le \sum_{i=1}^{n} b_i$, then a solution with sum exactly T can be found easily in $O(n)$ time. Start by setting $x_i = a_i$, $\forall i$. For $i = 1, 2, \ldots, n$, if we can increase x_i from a_i to b_i without violating the T bound, we set $x_i = b_i$. The first time we reach an index j when increasing x_j to b_j makes $\sum_{i=1}^{n} x_i$ larger than T, we stop and set $x_j = T - \sum_{i \neq j} x_i$. Thus, we assume that $T < \sum_{i=1}^{n} a_i$. In summary, from now on we only consider the non-trivial case $\max_i b_i < T < \sum_i a_i$.

Next we divide items into two types: *small* items and *large* items. Small items are those such that $a_i \le \varepsilon T$, large items are all others. The crucial observation here is that a relatively large a_i makes the approximation problem difficult, and it really doesn't matter what b_i is. The next lemma guarantees that we only need to consider large items for the approximation algorithm:

Lemma 2. *If we can compute an $(1+\varepsilon)$-approximation for all the large items, we can compute an $(1+\varepsilon)$-approximation for all the items with $O(n)$ extra time.*

According to Lemma 2, only large items need to be considered. Once we obtain a good solution for large items, small items can be added to it afterwards. So from now on we assume that all items are large. We are now ready to describe a $(1+\varepsilon)$-approximation algorithm with running time $O(n/\varepsilon^2)$. For simplicity of exposition, we assume that $1/\varepsilon$ is an integer for the rest of the paper. We exploit the structure of the canonical optimal solution and describe the algorithm in three steps.

First, we scale all interval endpoints by a factor $\varepsilon^2 T$. Specifically, let $a'_i = \lfloor a_i/(\varepsilon^2 T) \rfloor$, $b'_i = \lfloor b_i/(\varepsilon^2 T) \rfloor$, for each i. Thus $a'_i \le b'_i \le 1/\varepsilon^2$. Now we only consider subset sums for the scaled down values, with the upper bound $1/\varepsilon^2$.

Second, we use dynamic programming to build two sets of lists $L_a(i)$, $L_b(i)$ for $i = 1, \ldots, n$. List $L_a(i)$ consists of all possible subset sums ($\le 1/\varepsilon^2$) for values $a'_1, \ldots, a'_i$, sorted in strictly increasing order. List $L_b(i)$ consists of all possible subset sums ($\le 1/\varepsilon^2$) for values $b'_i, \ldots, b'_n$. We build list $L_a(i)$ for $i = 1, \ldots, n$ starting from $i = 1$. List $L_a(1)$ consists of only one single element a'_1. Once we get list $L_a(i-1)$, we build $L_a(i)$ as follows: Let L_{temp} be the list by adding a'_i into each element of $L_a(i-1)$, and discarding all values exceeding $1/\varepsilon^2$. L_{temp} is also a sorted list. Let $L_a(i)$ be the list by merging $L_a(i-1)$ and L_{temp}. Since each list has length bounded by $1/\varepsilon^2$, it takes $O(1/\varepsilon^2)$ time to build $L_a(i)$ from $L_a(i-1)$. $L_b(i)$ is constructed similarly; we start with $i = n$ and build $L_b(i)$ by using $L_b(i+1)$ and b'_i. There are totally $2n$ lists, which takes $O(n \cdot 1/\varepsilon^2)$ time and space to build.

Third, for each possible midrange index $r \in \{1, \ldots, n\}$, let $V(r)$ denote

$$\max\{v_1 + x'_r + v_2 \mid v_1 \in L_a(r-1),\ v_2 \in L_b(r+1),\ x'_r \in [a'_r, b'_r],\ v_1 + x'_r + v_2 \le 1/\varepsilon^2\}.$$

According to Lemma 1, we only need to consider canonical solutions of the form given by $V(r)$, for some midrange index r. In the following we claim that $V(r)$ can be computed in linear time:

Claim. We can compute $V(r)$ in $O(1/\varepsilon^2)$ time for a fixed r.

If the claim is true, then we can compute all the $V(r)$'s for $r = 1, \ldots, n$ in time $O(n \cdot 1/\varepsilon^2)$, and the output will be $\max_{1 \le r \le n} V(r)$. In the following we show how to prove the above claim.

Let v_1 and v_2 be generic entries in the lists $L_a(r-1)$ and $L_b(r+1)$ resp. We call a pair (v_1, v_2) feasible iff $v_1 + v_2 \le 1/\varepsilon^2 - a'_r$. It is easy to see that to compute $V(r)$ we only need to consider feasible pairs.

Next, recall that both of the lists, $L_a(r-1)$ and $L_b(r+1)$, are sorted in the ascending order. Now to compute $V(r)$ we are going to do a linear walk on these two lists: starting from the head of $L_a(r-1)$ and the tail of $L_b(r+1)$. Let $v_1 \in L_a(r-1)$ and $v_2 \in L_b(r+1)$ be the current elements we are considering. If (v_1, v_2) is not a feasible pair then we know that for any $v'_2 \ge v_2$, (v_1, v'_2) is not going to be feasible either. Therefore, we fix v_1 and reduce v_2 by moving one step left. If (v_1, v_2) is feasible, then we do some extra processing: If $v_1 + v_2 \le 1/\varepsilon^2 - b'_r$, there exists $x_r \in [a'_r, b'_r]$ such that $v_1 + v_2 + x_r = 1/\varepsilon^2$. In this case, we have already obtained the optimal solution and stop. Otherwise, we keep track of $v_1 + v_2 + b'_r$ as a candidate for $V(r)$. Now we increase v_1 by moving one step right. Once we reach the end of any of the list, we exit and choose the candidate solution with the maximum value. Given that we walk through these two lists monotonically, the total time spent to compute $V(r)$ will be bounded by a constant factor of the maximum list length, which is $O(1/\varepsilon^2)$. Now we're ready to state our first theorem:

Theorem 2. *The above algorithm computes a* $(1+\varepsilon)$*-approximation of ISSP in time and space* $O(n \cdot 1/\varepsilon^2)$.

3.2 An Improved FPTAS

In this subsection we give an algorithm with improved time and space complexity. Recall that we scale down all the intervals in the previous subsection. Scaling will cause an error for each item value, thus the total aggregated error is large afterwards. To get a better performance, we avoid scaling and use a technique called *relaxed dynamic programming* (relaxed-DP). Relaxed-DP avoids scaling and keeps the total error bounded by εT during the DP process. The basic idea of relaxed-DP is based on Kellerer et al. [8].

Partition the space $(0, T]$ into $1/\varepsilon$ intervals $B_j = ((j-1) \cdot \varepsilon T, j \cdot \varepsilon T]$ for $j = 1, \ldots, 1/\varepsilon$. The relaxed-DP will build two DP tables $L_a(i, j)$ and $L_b(i, j)$ for $i = 0, \ldots, n$ and $j = 1, \ldots, 1/\varepsilon$. Let's describe $L_a(i, j)$ in detail and $L_b(i, j)$ is built similarly.

For values $a_1, \ldots, a_i$, there may be a lot of feasible subset sums falling into the range B_j. To reduce time and space complexity, the algorithm will only keep at most two possible values and store them in $L_a(i, j)$. $L_a(i, j)$ will either be empty, or contain two of the possible subset sums v^-_{ij}, v^+_{ij} with $v^-_{ij} \le v^+_{ij}$. Intuitively, v^-_{ij} denotes one of the smallest subset sums in the interval B_j and v^+_{ij} denotes one of the largest subset sums in B_j. Initially $L_a(0, j)$ is empty for all j. For the dynamic programming step from $i-1$ to i, let's consider item a_i.

Let $\tilde{\Delta}_i$ be all the possible values of $(a_i + v^-_{i-1,j})$, $(a_i + v^+_{i-1,j})$ in $(0, T]$ for $j = 1, \ldots, 1/\varepsilon$. The cardinality of $\tilde{\Delta}_i$ is $\leq 2/\varepsilon$. For each possible value $v \in \tilde{\Delta}_i$, find the corresponding interval B_j that it belongs to. If $L_a(i, j)$ is empty, then set $v^-_{ij} = v^+_{ij} = v$. Otherwise, update $L_a(i, j)$ as the following:

$$v^-_{ij} := \min\{v^-_{ij}, v\}, \quad v^+_{ij} := \max\{v^+_{ij}, v\}.$$

It is easy to check that the step from $i-1$ to i takes time $O(1/\varepsilon)$ and the whole process to build $L_a(i, j)$ for all i, j can be done in time and space $O(n \cdot 1/\varepsilon)$.

For a fixed i, consider all the values v^-_{ij}, $v^+_{i,j}$ for $j = 1, \ldots, 1/\varepsilon$. There are totally at most $2/\varepsilon$ of them and we relabel them as an ordered list $v^a_i(1) < \cdots < v^a_i(\lambda_i)$, where $\lambda_i \leq 2/\varepsilon$ is the number of distinct values in $L_a(i, .)$. For convenience, we also use $v^a_i(.)$ to denote the set containing all these λ_i elements. Let Δ_i denote all the possible subset sums for elements $a_1, \cdots, a_i$. The following lemma shows that the sequence $v^a_i(.)$ is an (εT)-approximation of Δ_i.

Lemma 3. *Let $v^a_i(0) = v^a_i(1) - \varepsilon T$, $v^a_i(\lambda_i + 1) = v^a_i(\lambda_i) + \varepsilon T$. For each possible subset sum $\delta \in \Delta_i$, there exists an index λ with $v^a_i(\lambda) \leq \delta \leq v^a_i(\lambda + 1)$ and $v^a_i(\lambda + 1) - v^a_i(\lambda) \leq \varepsilon T$.*

The proof can be found in Kellerer et al. [8]. Similarly, $L_b(i, j)$ is either empty or contains at most two feasible subset sums over $\{b_i, \ldots, b_n\}$ in the range B_j. As before, we can compute all the values $L_b(i, j)$ in time $O(n \cdot 1/\varepsilon)$. And $v^b_i(.)$ is an (εT)-approximation for the subset sums over $b_i, \ldots, b_n$. We are now ready to compute the final solution based on the canonical structure of the optimal solution. Similar as in Section 3.1, for each possible midrange index r, let $V(r)$ denote

$$\max\{v_1 + x_r + v_2 \mid v_1 \in v^a_{r-1}(.),\ x_r \in [a_r, b_r],\ v_2 \in v^b_{r+1}(.),\ v_1 + x_r + v_2 \leq T\}.$$

We claim that $V(r)$ can be computed in $O(1/\varepsilon)$ time for a fixed r. The claim can be proved identically as Claim 3.1, with the only superficial difference that here $v^a_{r-1}(.), v^b_{r+1}(.)$ have their cardinalities bounded by $2/\varepsilon$, while in Claim 3.1 $L_a(r-1), L_b(r+1)$ have their cardinalities bounded by $1/\varepsilon^2$. Because the claim is true, we can compute all the $V(r)$'s for $r = 1, \ldots, n$ in time $O(n \cdot 1/\varepsilon)$. The output of the algorithm will be $\max_{1 \leq r \leq n} V(r)$. Now we're ready to state our main result in this section:

Theorem 3. *We can compute a $(1 + \varepsilon)$-approximation of ISSP in time and space $O(n \cdot 1/\varepsilon)$.*

4 Approximation Schemes for kISSP

In the cardinality constrained interval subset-sum problem (kISSP), two additional integers $k_{\min}$ and $k_{\max}$ are specified. The goal is to reach (approximate) the target sum T using at least $k_{\min}$ and at most $k_{\max}$ (non-zero) elements. With this additional constraint, even finding a simple 2-approximation is not trivial. The following lemma describes such an algorithm.

Lemma 4. *A 2-approximation of kISSP can be computed in $O(n)$ time and space.*

The fully polynomial time approximation scheme for kISSP is similar to the improved FPTAS described in Section 3.2 for ISSP, as it also uses relaxed-DP to improve its performance. Partition the space $(0, T]$ into $1/\varepsilon$ intervals $B_j = ((j-1)\cdot\varepsilon T, j\cdot\varepsilon T]$ for $j = 1, \ldots, 1/\varepsilon$. The relaxed-DP technique will build two DP tables $L_a(i, j, k)$ and $L_b(i, j, k)$ for indexes $i = 0, \ldots, n$, $j = 1, \ldots, 1/\varepsilon$, and $k = 1, \ldots, k_{\max}$. Let's describe $L_a(i, j, k)$ in detail and $L_b(i, j, k)$ is built similarly.

Fix i, j, k as in their specified domains, $L_a(i, j, k)$ will either be empty or contains two possible subset sums over $\{a_1, \ldots, a_i\}$ in the range B_j. These subset sums should be the summation of exactly k (non-zero) values over $a_1, \ldots, a_i$. If $L_a(i, j, k) \neq \emptyset$, then it contains v^-_{ijk}, v^+_{ijk} with $v^-_{ijk} \leq v^+_{ijk}$. Intuitively, v^-_{ijk} denotes one of the smallest subset sums in the interval B_j with cardinality k, and v^+_{ijk} denotes one of the largest subset sums in B_j with cardinality k. By definition, $L_a(i, j, k)$ is empty if $i < k$ as the total number of available values is less than k. And $L_a(0, j, k)$ is empty for any j, k. For the dynamic programming step from $i-1$ to i, let's consider item a_i.

For fixed i, k, let $\tilde{\Delta}_{ik}$ be all the possible values of $(a_i + v^-_{i-1,j,k-1})$, $(a_i + v^+_{i-1,j,k-1})$ for j from 1 to $1/\varepsilon$. There are at most $2/\varepsilon$ of them. For each possible value $v \in \tilde{\Delta}_{ik}$, find the corresponding interval B_j that it belongs to. If $L_a(i, j, k)$ is empty, then set $v^-_{ijk} = v^+_{ijk} = v$. Otherwise, update $L_a(i, j, k)$ as the following:

$$v^-_{ijk} := \min\{v^-_{ijk}, v\}, \quad v^+_{ijk} := \max\{v^+_{ijk}, v\}.$$

It is easy to check that the step from $i-1$ to i takes time $O(k_{\max} \cdot 1/\varepsilon)$ and the whole process to build $L_a(i, j, k)$ for all i, j, k can be done in time and space $O(nk_{\max} \cdot 1/\varepsilon)$.

For fixed i, k, we consider all the values v^-_{ijk}, v^+_{ijk} for $j = 1, \ldots, 1/\varepsilon$. There are totally $\lambda_{ik} \leq 2/\varepsilon$ distinct elements and we relabel them as $v^a_{ik}(1) < \cdots < v^a_{ik}(\lambda_{ik})$. For convenience, we will also use $v^a_{ik}(.)$ to denote the set containing all these elements. Let Δ_{ik} denote all the possible subset sums (with fixed cardinality k) of $a_1, \cdots, a_i$. The following lemma shows that $v^a_{ik}(.)$ is an (εT)-approximation for Δ_{ik}:

Lemma 5. *Let $v^a_{ik}(0) = v^a_{ik}(1) - \varepsilon T$, $v^a_{ik}(\lambda_{ik} + 1) = v^a_{ik}(\lambda_{ik}) + \varepsilon T$. For each possible subset sum $\delta \in \Delta_{ik}$, there exists an index j with $v^a_{ik}(j) \leq \delta \leq v^a_{ik}(j+1)$ and $v^a_{ik}(j+1) - v^a_{ik}(j) \leq \varepsilon T$.*

Similarly, we define and compute $L_b(i, j, k)$ for all i, j, k and $v^b_{ik}(.)$ is an (εT)-approximation for all subset sums over $\{b_i, \ldots, b_n\}$ with cardinality k. We are now ready to compute the final solution based on the canonical structure of the optimal solution. As in Section 3.2, we can define $V(r, k_1, k_2)$ as the maximum solution value with the form that $v_1 \in v^a_{r-1,k_1}(.)$, and $v_2 \in v^b_{r+1,k_2}(.)$. This straightforward implementation results in a multiplicative factor of $k^2_{\max}$ for the time complexity. A relatively involved technique can reduce the multiplicative

factor from $k_{\max}^2$ to $k_{\max}$, thus *matches* the time complexity of the best algorithm for cardinality constrained subset sum problem. For details about how to improve the time factor from $O(k_{\max}^2)$ to $O(k_{\max})$, see the proof of the following theorem, which is also our main result in this section.

Theorem 4. *We can compute a* $(1+\varepsilon)$*-approximation of kISSP in time and space* $O(n \cdot k_{\max} \cdot 1/\varepsilon)$.

5 Application to Uniform-Price Auction Clearing

In this section we reduce the auction clearing problem for uniform-price multi-unit auctions to ISSP (or kISSP), and use algorithms for ISSP (or kISSP) to solve the auction clearing problem.

For forward auctions, assume that there are T units of a single type of goods and the seller wants to maximize his revenue. There are n bidders, and the i-th bidder submits a bid $(p_i, [a_i, b_i])$ where p_i denotes the unit price and $[a_i, b_i]$ is the desired quantity range. The clearing algorithm works as follows: For each index i, find all the intervals $S_i = \{[a_{i'}, b_{i'}] \mid p_i \leq p_{i'}\}$. S_i makes an instance of ISSP (or kISSP) with target T, and we apply FPTAS for ISSP (or kISSP) to compute the maximum value V_i. Find index i such that $p_i V_i$ is maximized over all the indexes, and p_i will the clearing price. In summary, we have:

Corollary 1. *For uniform-price multi-unit auctions with n bidders and each bid has an interval quantity range, we can compute a* $(1+\varepsilon)$*-approximation in time* $O(n^2 \cdot 1/\varepsilon)$. *If cardinality constraints are present, a* $(1+\varepsilon)$*-approximation can be obtained in time* $O(n^2 \cdot k_{\max} \cdot 1/\varepsilon)$.

For procurement auctions with the piecewise constant supply curve and uniform-price model, similar results can be obtained. Observe that the uniform price offered to all the winning suppliers is going to be one of the price levels in the input functions. For each price level, we solve an instance of ISSP as follows: Consider a unit price p. Let $a_i(p)$ (resp. $b_i(p)$) denote the minimum (resp. maximum) number of units offered by i at price p or lower. These n intervals $[a_i(p), b_i(p)]$ form the input to ISSP, with target T. Since all winning suppliers are paid the common price p, the total cost becomes $pT(p)$, where $T(p)$ is the solution of the corresponding ISSP. Thus we should choose a price level p^* such that $p^*T(p^*)$ is minimized. Therefore the procurement problem can be solved by solving m instances of ISSP, where m is the total number of different price levels in the input. When the number of suppliers must be bounded in the range $[k_{\min}, k_{\max}]$, we use kISSP. In summary, we have

Corollary 2. *For procurement auctions with uniform-price rule and piecewise constant supply curves for suppliers, we can compute a* $(1+\varepsilon)$*-approximation in time* $O(m^2 \cdot 1/\varepsilon)$. *Here m is the total number of different price levels. If cardinality constraints are present, a* $(1+\varepsilon)$*-approximation can be obtained in time* $O(m^2 \cdot k_{\max} \cdot 1/\varepsilon)$.

6 Concluding Remarks

We gave a fully polynomial time approximation scheme for the interval subset sum problem, a generalization of the classical subset sum problem. We also presented a FPTAS to solve the cardinality constrained version of the problem. Our algorithms for both ISSP and kISSP have the same asymptotic complexity as the corresponding classical versions of the subset sum problem. The interval subset-sum problem extends the subset-sum problem to intervals, and it finds a natural application in uniform-price multi-unit auctions. With the advance of auction design mechanism and practice, we foresee that bidder's single quantity bidding format will possibly be replaced by an interval quantity format. Under such circumstance, it is likely for the interval subset sum algorithm to become a useful subroutine for future auction clearing algorithms.

Due to space limit, we didn't consider how to improve space complexity. It is possible to improve the space bounds for our algorithms using standard techniques such as balanced dynamic programming. Currently the auction clearing algorithm invokes the ISSP (or kISSP) procedure $O(n)$ times, so that its time complexity is quadratic of n. Given that these ISSP (or kISSP) instances are almost identical, is it possible to reduce the time complexity from $O(n^2)$ to sub-quadratic, such as $O(n \log n)$? We leave this as an open problem.

References

1. A. Caprara, H. Kellerer, U. Pferschy and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:333–345, 2000.
2. A. Dailianas, J. Sairamesh, V. Gottemukkala and A. Jhingran. Profit-driven matching in e-marketplaces: Trading composable commodities. *Agent Mediated Electronic Commerce Workshop*, 153–179, 1999.
3. A. Davenport, J. Kalagnanam and H.S. Lee. Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. *Electronic Commerce Research*, 1(3):221–238, 2001.
4. M. Eso et al. Bid evaluation in procurement auctions with piece-wise linear supply curves. Technical Report RC 22219, IBM Research, 2001.
5. P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849-859, 1961.
6. O. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *JACM*, 22:463–468, 1975.
7. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, 43:85–103, 1972.
8. H. Kellerer, R. Mansini, U. Pferschy and M.G. Speranza. An efficient fully polynomial approximation scheme for the Subset-Sum Problem. *Journal of Computer and System Science*, 66:349–370, 2003.
9. H. Kellerer, U. Pferschy and D. Pisinger. *Knapsack Problems*. Springer, 2004.
10. F. Kelly and R. Steinberg. A combinatorial auction with multiple winners for universal services. *Management Science*, 46:586–596, 2000.
11. A. Kothari, D. Parkes and S. Suri. Approximately-strategyproof and tractable multi-unit auctions. *Decision Support Systems*, 39(1):105-121, 2005.

12. N. Nisan and A. Ronen. Algorithmic mechanism design. *STOC*, 129–140, 1999.
13. L. I. O'Callaghan, D. Lehmann and Y. Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. *Proc. of ACM EC*, 96–102, 1999.
14. A. Pekec, M. H. Rothkopf and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44:1131–1147, 1998.
15. D. Pisinger. Linear time algorithms for knapsack problems with bounded weight. *Journal of Algorithms*, 33:1–14, 1999.
16. M. P. Wellman, W. E. Walsh and F. Ygge. Combinatorial auctions for supply chain formation. *Proc. of ACM EC*, 260–269, 2000.
17. Google IPO Prospectus. https://www.ipo.google.com/data/prospectus.html.

Improved Algorithms for the K-Maximum Subarray Problem for Small K

Sung E. Bae and Tadao Takaoka

Department of Computer Science and Software Engineering
University of Canterbury, Christchurch, New Zealand
{seb43,tad}@cosc.canterbury.ac.nz

Abstract. The maximum subarray problem for a one- or two-dimensional array is to find the array portion that maiximizes the sum of array elements in it. The K-maximum subarray problem is to find the K subarrays with largest sums. We improve the time complexity for the one-dimensional case from $O(min\{K + n\log^2 n, n\sqrt{K}\})$ for $0 \leq K \leq n(n-1)/2$ to $O(n \log K + K^2)$ for $K \leq n$. The latter is better when $K \leq \sqrt{n}\log n$. If we simply extend this result to the two-dimensional case, we will have the complexity of $O(n^3 \log K + K^2 n^2)$. We improve this complexity to $O(n^3)$ for $K \leq \sqrt{n}$.

1 Introduction

The maximum subarray problem was first described by Bentley in his literature *Programming Pearls* [4, 5] as an example to discuss the efficiency of computer programs. This problem determines an array portion that sums to the maximum value with respect to all possible array portions within the input array. When the input array is two-dimensional, we find a rectangular subarray with the largest possible sum.

If all elements of an array are non-negative, this problem is trivial, as the entire array represents the solution. Similarly, if all elements are non-positive, the solution is empty with value 0. So we consider a data set containing both positive and negative values. In practice, a bitmap image has all non-negative pixel values. When the average is subtracted from each pixel, we can apply the maximum subarray algorithm to find the brightest area within the image.

For the one-dimensional case, we have an optimal linear time sequential solution. A simple extension of this solution can solve the two-dimensional problem in $O(m^2 n)$ time for an $m \times n$ array ($m \leq n$), which is cubic when $m = n$ [4, 5]. In this paper, if only n appears in complexities for the two-dimensional case, we assume $m = n$. The sub-cubic time solution based on Takaoka's sub-cubic *distance matrix multiplication algorithm* [14] is given by Tamaki and Tokuyama [17], which is further simplified by Takaoka [15]. In the context of parallel computations, time and cost optimal PRAM and mesh algorithms for the one-dimensional case are described in [10]. For the two-dimensional case, EREW PRAM solutions achieving $O(\log n)$ time with $O(n^3/\log n)$ processors are given in [11, 18] and comparable result on interconnection networks is given in [12]. The EREW PRAM version of the subcubic algorithm in [15, 17] is given in [1], which also features a VLSI algorithm based on the technique introduced

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 621–631, 2005.

in Bentley's paper. This VLSI algorithm for the maximum subarray problem achieves $T = m + n - 2$ steps, which is $O(n)$ time using $O(n^2)$ sized hardware circuit.

Finding K maximum sums is a natural extension. This problem is discussed in [2] and [3]. The former provides $O(Kn)$ and $O(Km^2n)$ time solutions for the one- and two-dimensional cases in the course of development of a systolic array algorithm of $O(n)$ time using $O(n^2)$ size hardware for the two-dimensional case. The latter brings the worst case time down to $O(min\{K+n\log^2 n, n\sqrt{K}\})$ for a one-dimensional array.

This paper reviews the former solution and tunes it up for greater speed. Specifically we achieve $O(n \log K + K^2)$ time for the one-dimensional case. This is better than [3] when $K \leq \sqrt{n}\log n$.

If we use the above algorithm directly for the two-dimensional maximum subarray problem with an (n, n)-array, we have $O(n^3 \log K + n^2K^2)$ time complexity. We improve this time complexity to $O(n^3 + n^2K^2)$, which is $O(n^3)$ when $K \leq \sqrt{n}$.

A related topic is a similar problem with K disjoint subarrays, which may be more practical in some applications. Within this category, we can define several problems, and only the one-dimensional case received some attention, especially in bio-informatics. Further discussion on a possible extension will be made in the section of concluding remarks.

2 Review of the Maximum Subarray Problem

We give a two-dimensional array $a[1..m, 1..n]$ as input data set. The maximum subarray problem is to find a rectangular portion $a[r_1..r_2, c_1..c_2]$ such that the sum of contained elements should be greater than or equal to the sum of any other rectangular portions of the data set. We suppose the upper-left corner has coordinates (1,1).

Example 1. : Let a be given by

$$a = \begin{bmatrix} -1 & 2 & -3 & 5 & -4 & -8 & 3 & -3 \\ 2 & -4 & -6 & -8 & 2 & -5 & 4 & 1 \\ 3 & -2 & 9 & -9 & \begin{bmatrix} -1 & 10 \\ 8 & -2 \end{bmatrix} & & -5 & 2 \\ 1 & -3 & 5 & -7 & & & 2 & -6 \end{bmatrix}$$

The maximum subarray is the array portion $a[3..4, 5..6]$ surrounded by inner brackets, whose sum is 15.

Bentley introduced Kadane's algorithm that finds the maximum sum within a one-dimensional array, whose time is linear [4], and extended it to two-dimensions.

We use another $O(n)$ algorithm given in [2]. It has its central algorithmic concept in the prefix sum. The prefix sums $sum[1..n]$ of a one-dimensional array $a[1..n]$ are computed by

Algorithm 1 Prefix Sum

```
sum[0]←0;
for i←1 to n do sum[i]←sum[i-1]+a[i];
```

As $sum[x] = \sum_{i=1}^{x} a[i]$, the sum of $a[x..y]$ is computed by the subtraction of these prefix sums such as:

$$\sum_{i=x}^{y} a[i] = sum[y] - sum[x-1]$$

To yield the maximum sum from a one-dimensional array, we have to find indices x, y that maximize $\sum_{i=x}^{y} a[i]$. The notations min and max are used for variables and MAX and MIN are used for operations.

Let min_i be the minimum prefix sum for an array portion $a[1..i-1]$. Then the following lemma is obvious.

Lemma 1. *For all* $x, y \in [1..n]$, *and* $x \leq y$,
$\mathrm{MAX}_{1\leq x\leq y\leq n}\{\sum_{i=x}^{y} a[i]\} = \mathrm{MAX}_{1\leq x\leq y\leq n}\{sum[y] - sum[x-1]\}$
$= \mathrm{MAX}_{1\leq y\leq n}\{sum[y] - \mathrm{MIN}_{1\leq x\leq y}\{sum[x-1]\}\} = \mathrm{MAX}_{1\leq y\leq n}\{sum[y] - min_y\}$

Based on Lemma 1, we can design the following linear time algorithm that finds the maximum sum in a one-dimensional array. Comments are given by //.

Algorithm 2 Maximum Sum in a one-dimensional array

```
min←0; //minimum prefix sum
M←0; //current maximum sum, initially 0 for empty subarray
sum[0]←0;
for i←1 to n do begin
    sum[i]←sum[i-1]+a[i];
    cand←sum[i]-min; //min=min_i
    M←MAX{M,cand};
    min←MIN{min,sum[i]};//min=min_{i+1}
end.
```

While we accumulate $sum[i]$, the prefix sum, we also maintain min, the minimum of the preceding prefix sums. By subtracting min from $sum[i]$, we have a candidate for the maximum sum. At the end, M is the maximum sum.

3 Finding the K Maximum Sums in $O(Kn)$ Time

Based on the algorithm in Section 2, let us proceed to discuss the K-maximum subarray problem, again for the one-dimensional case.

Instead of having a single variable that book-keeps the minimum prefix sum, we maintain a list of K minimum prefix sums, sorted in non-decreasing order.

Let min_i be the list of K minimum prefix sums for $a[1..i-1]$ given by $\{min_i[1]\cdots, min_i[K]\}$, sorted in non-decreasing order. The initial value for min_i is given by $min = \{0, +\infty\cdots, +\infty\}$. We also maintain the list of candidate sums produced from $sum[i]$, sorted in non-decreasing order. This list $cand_i$ is given by $\{sum[i]-min_i[1], sum[i]-min_i[2]\cdots, sum[i]-min_i[K]\}$. Let max_i be the list of K maximum sums for $a[1..i]$. This list is maintained in M in Algorithms 3 and 4 sorted in non-increasing order. When the algorithm ends, M contains the final solution max_n. The merged list of two sorted sequences x and y are denoted by $merge(x, y)$. We have the following lemma.

Lemma 2. max_{i+1} is the list of the K maximum elements of $merge(max_i, cand_{i+1})$

Array names are used to denote sets, lists, etc. in the subsequent descriptions. We maintain the list of K minimum prefix sums in min. Each time a prefix sum is computed, we subtract these K minima from this prefix sum, and prepare a list $cand$ of

candidate K maximum values. These K values are merged with the current maximum sums stored in M, from which we choose the largest K values. After this, we insert the prefix sum to the list of K minimum prefix sums for the next iteration. When a new entry is inserted, the list of K minimum prefix sums has $K+1$ items. By discarding the largest one, we keep the size of this list to be fixed at K. Of course, if this sum is found to be greater than all current K minima, no insertion is made.

Note that we initialize the list of tentative solutions by $M = \{0, -\infty \cdots, -\infty\}$.

The line 8 in the algorithm preserves the loop-invariant from step i to step $i+1$ as stated in Lemma 2. At the end, M is the solution.

Algorithm 3 K maximum sums in a one-dimensional array

```
 1: for k←1 to K do begin
 2:     min[k]←∞; M[k]←−∞;
 3: end;
 4: sum[0]←0; min[1]←0; M[1]←0;
 5: for i←1 to n do begin
 6:     sum[i]←sum[i-1]+a[i];
 7:     for k←1 to K do cand[k]←sum[i]-min[k];
 8:     M←K largest elements of merge(M,cand);
 9:     insert sum[i] into min;
10: end.
```

At each iteration, it takes $O(K)$ time for generating the candidate list, and $O(K)$ time for merging this list and the list of current maximum sums. Inserting a prefix sum into the list of minimum prefix sums depends on what data structure is used. If it is a simple array or list, the insertion takes $O(K)$ time, which establishes $O(K)$ overall time for each iteration. Using an advanced data structure makes little significance at this point due to line 7 and 8 where we anyway need to spend $O(K)$ time generating the candidate list and the list of K maximum sums at each iteration.

As we need to perform n iterations, the total time complexity is $O(Kn)$. When $K = 1$, this result is comparable to $O(n)$ time of Kadane's algorithm and Algorithm 2.

4 Improved Algorithm for K Maximum Sums for Small K

Previously, we generated the list of candidates by subtracting the K minimum prefix sums from each prefix sum, which results in production of Kn candidates in total. K maximum sums are basically selected from this pool of Kn candidates. It will now be discussed that we do not need to generate such a number of candidates when $K \leq n$.

Let us assume we have in $min_i[1..K]$ the list of K minimum prefix sums to be subtracted from $sum[i]$. This list is sorted in non-decreasing order. In Algorithm 4, min_i is given by min at the end of the i-th iteration.

Let $cand_i[k] = sum[i] - min_i[k]$ for $k = 1 \cdots, K$. As min_i is sorted, the produced list of candidates $cand_i$ is sorted in non-increasing order, and has the first item $cand_i[1]$ being the largest candidate produced from $sum[i]$.

We first produce n samples of $cand_1[1] \cdots, cand_n[1]$ and let them be elements of a list $sample$.

$$sample[i] = cand_i[1] = sum[i] - min_i[1], (i = 1 \cdots, n)$$

We then select K largest values of $sample$. Let us denote the list containing them by $Ksamples$, which is sorted in non-increasing order, given by

$$Ksamples = \{sample[x_1], sample[x_2] \cdots, sample[x_K]\},$$

where $x_1 \cdots, x_K$ are the indices of those selected samples.

It is easily observed that if $sample[w]$, the largest candidate produced from $sum[w]$, does not even qualify for $Ksamples$, no candidate produced from $sum[w]$ can become one of the final K maximum sums as we know there are already at least K sums greater than or equal to them.

When $Ksamples$ does not include $sample[w]$, the full generation of $cand_w[1..K]$ is thus avoided. In such a case, we can skip to the next iteration saving $O(K)$ time. We generate candidates only from $sum[x_i]$, which produced selected candidates for $Ksamples$.

4.1 Pre-process

We note that we do not need $min_i[1..K]$ for all $i \in [1..n]$ before the sampling and selection process. We only need $min_i[1]$ for $i = 1 \cdots, n$.

During the pre-process, we traverse the input array $a[1..n]$ and compute the prefix sum $sum[1..n]$ in $O(n)$ time. Within this time frame, we find the minimum prefix sum ($min_i[1]$ only) for each $sum[i]$, as $min_i[1]$ is the minimum of $sum[j]$ for $1 \leq j \leq i-1$. Full lists of K minimum prefix sums for each $sum[i]$ are not produced during this pre-process.

The K-th maximum of this sample is selected by a linear time selection algorithm. Then we filter out values smaller than the K-th maximum, being left with the K largest samples. We sort and store those samples in $Ksamples$. We can test whether an item is in $Ksamples$ by comparing it with $sample[x_K]$, the last element in the list.

4.2 Candidate Generation and Selection

Inside the "for" loop starting at line 10 there are two parts, Part I and Part II. We consider time for each part separately.

Part I is for the generation of $cand_i$ and maintaining the tentative solution set M. The generation of $cand_i$ is performed when the i-th sample is included in $Ksamples$. Thus Part I is performed K times.

The following routine, Part II, is the insertion of prefix sum into the sorted list of minimum prefix sums. Unlike Part I, all n prefix sums should be considered. We first examine if a new prefix sum ever needs to be inserted, and if so we need to find an appropriate position for the new entry in min. This min contains K minimum prefix sums, and if there are more than K items, we may need to drop the largest item. The choice of an appropriate data structure for min is important to determine the total complexity. Besides min, all other lists, $cand$ and M, may be simple one-dimensional arrays. We assume $min[k]$ is the k-th smallest element of min when min is regarded as a set regardless of the actual data structure of min. We choose a 2-3 tree with *level-link* as a suitable data structure.

A 2-3 tree keeps all the leaf nodes sorted. A 2-3 tree with level-link described in [7] has all the internal nodes at the same depth connected, enabling *finger* searches. Finger search trees with constant update time are discussed in [6, 9], but they both require logarithmic time for positioning and do not improve overall time complexity. Now we analyze each part.

Part I. For Part I, generating the candidate list involves access to the list of minimum prefix sums. If an ordinary 2-3 tree is used, accessing each of $min[1..K]$ costs $O(\log K)$ time. Since we need to access all $min[1..K]$ sequentially to generate candidates, this access cost seems rather expensive. However if a 2-3 tree with level-link is used, after initial search for $min[1]$ spending $O(\log K)$, subsequent elements are found in $O(1)$ time due to finger search. As actual generation of K candidates requires $O(K)$ time, this initial $O(\log K)$ access time is absorbed. The total time for Part I over K iterations is therefore $O(K^2)$.

Part II. For Part II, finding position for a new entry and actual insertion is done in $O(\log K)$ time. When there are more than K items, deletion of the largest item and update of the tree costs another $O(\log K)$ time. For n iterations, the total time for Part II is $O(n \log K)$.

Algorithm 4 Improved algorithm for K maximum sums in a one-dimensional array

```
//[INITIALIZATION]
 1: for k←1 to K do begin min[k]←∞; M[k]←−∞; end;
 2: sum[0]←0; min[1]←0; M[1]←0;
//[PRE-PROCESS]
 3: for i←1 to n do begin
 4:     sum[i]←sum[i-1]+a[i];
  //sample for initial K large values
 5:     sample[i]←sum[i]-min[1];
 6:     if sum[i] < min[1] then min[1]←sum[i];
 7: end;
 8: Ksamples←K largest sorted values of sample[1..n];
//[CANDIDATE GENERATION and SELECTION]
 9: min[1]←0;
10: for i←1 to n do begin
11:     if sum[i]-min[1] > sample[x_K] then begin
            //PART I: Generate cand and update M
12:         for k←1 to K do cand[k]←sum[i]-min[k];
13:         M←K largest values of merge(M,cand);
14:     end;
        //PART II: Update min
15:     insert sum[i] into min;
16: end.
```

4.3 Total Time

Using the data structure for min described above, the overall time including Part I and Part II is thus $O(n \log K + K^2)$. The time for the preprocessing (sampling, selection and screening) is $O(n)$ which is absorbed. Compared with $O(min\{K + n \log^2 n, n\sqrt{K}\})$ time by [3], this algorithm is faster when $K \leq \sqrt{n} \log n$.

We can organize the if-statement at line 11 into a while-statement. We keep computing candidates as $sum[i] - min[k]$ for $k = 1, 2 \cdots, K$, while the condition $sum[i] - min[k] > sample[x_K]$ is satisfied, and only those candidates can be inserted into M with unqualified sums being deleted from M. Also the right-hand side of the condition can be replaced by the minimum of M instead of the fixed $sample[x_K]$. This modification can improve the average performance, but the worst case behavior is not clear at present.

5 Speed-Up for Two Dimensions

If we use the algorithm in the previous section for an (n, n)-array, we have an $O(n^3 \log K)$ time algorithm. We speed up the algorithm for small K in the two-dimensional case based on the divide-and-conquer method. To remove the factor of $\log K$ from the complexity, we do not maintain sorted order for K-tuples.

5.1 Generalization of Distance Matrix Multiplication

The distance matrix multiplication is to compute the following distance product $C = AB$ for two (n, n)-matrices $A = [a_{ij}]$ and $B = [b_{ij}]$ whose elements are real numbers.

$$c_{ij} = \mathrm{MIN}_{k=1}^{n}\{a_{ik} + b_{kj}\}, (i, j = 1 \cdots, n) \cdots (1)$$

The operation in the right-hand side of (1) is called distance matrix multiplication of MIN-version, and A and B are called distance matrices in this context. If we use MAX instead we call it the MAX-version.

Now we divide A, B, and C into (K, K)-submatrices for $N = n/K$ as follows:

$$\begin{pmatrix} A_{1,1} & \cdots & A_{1,N} \\ \cdots & & \\ A_{N,1} & \cdots & A_{N,N} \end{pmatrix} \begin{pmatrix} B_{1,1} & \cdots & B_{1,N} \\ \cdots & & \\ B_{N,1} & \cdots & B_{N,N} \end{pmatrix} = \begin{pmatrix} C_{1,1} & \cdots & C_{1,N} \\ \cdots & & \\ C_{N,1} & \cdots & C_{N,N} \end{pmatrix}$$

Matrix C can be computed by

$$C_{ij} = \mathrm{MIN}_{k=1}^{N}\{A_{ik}B_{kj}\}(i, j = 1 \cdots, N) \cdots (2)$$

where the product of submatrices is defined similarly to (1) and the MIN operation is defined on submatrices by taking the MIN operation component-wise. Since comparisons and additions of distances are performed in a pair, we omit counting the number of additions for measurement of the complexity. We have N^3 multiplications of distance matrices in (2).

To prepare for the K-maximum subarray problem, we extend equation (1) in such a way that c_{ij} is the K-tuple of K minima of $\{a_{ik} + b_{kj} | k = 1 \cdots, n\}$. We call this definition K-distance matrix multiplication, or simply K-matrix multiplication.

Now we generalize the MIN and MAX operations on distance matrices. Let each element of a distance matrix be a K-tuple of real numbers such as $\mathbf{a} = (a_1 \cdots, a_K)$. The MIN operation on the two K-tuples $\mathbf{a}$ and $\mathbf{b}$ is defined by $\mathrm{MIN}\{\mathbf{a}, \mathbf{b}\} = (c_1 \cdots, c_K)$, where $(c_1 \cdots, c_K)$ is the list of the K smallest elements of $\mathbf{a} \cup \mathbf{b}$. If there are equal values in $\mathbf{a}$ or $\mathbf{b}$, the union operation here is for multisets. Similarly we can define $\mathrm{MAX}\{\mathbf{a}, \mathbf{b}\} = \mathbf{a} \cup \mathbf{b} - (c_1 \cdots, c_K)$. The extended MIN and MAX operations can be performed by taking the smaller half and larger half from $\mathbf{a} \cup \mathbf{b}$, which can be done in $O(K)$ time with the median selection algorithm and filtering process in a similar way to those described in Section 4.1. In the following we mainly describe the MIN-version. The MAX-version can be defined symmetrically.

If each element of distance matrices A_1 and A_2 is a K-tuple, the MIN operation on A_1 and A_2 is defined component-wise over corresponding K-tuples. To compute K-matrix multiplication, where each element in (1) is a K-tuple, we use the extended MIN operation in (2), where the elements of matrix $A_{ik}B_{kj}$ are K-tuples.

Let us rename A_{ik} and B_{kj} in the above by A and B, and consider the multiplication. This time we can return all $\{a_{i1} + b_{1j} \cdots, a_{iK} + b_{Kj}\}$ as candidate K-tuples, taking $O(K^3)$ time, and use the extended MIN operations in (2). Then the time for N^3 products in (2) is $O((n/K)^3 K^3) = O(n^3)$. The time for extended MIN operations in (2) is $O(Nn^2K) = O(n^3)$. Thus the total time is $O(n^3)$ for K-matrix multiplication.

5.2 Application to the K-Maximum Subarray Problem

We review the divide-and-conquer approach given in [15]. Let a two-dimensional array $a[1..m, 1..n]$ of real numbers be given as input data. The maximum subarray problem here is to maximize the sum of the array portion $a[k..i, l..j]$, that is, to obtain such indices (k, l) and (i, j).

We assume that $m \leq n$ without loss of generality. We also assume that m and n are powers of 2. We will mention the general case of m and n later.

The central algorithmic concept in this section is again that of prefix sum. We use distance matrix multiplications of both MIN and MAX versions in this section. We compute the prefix sums $s[i, j]$ for array portions of $a[1..i, 1..j]$ for all i and j with the boundary condition $s[i, 0] = s[0, j] = 0$. Obviously this can be done in $O(mn)$ time. The outer framework of the algorithm is given below. Note that the prefix sums once computed are used throughout recursion.

Algorithm M: Maximum subarray
1. If the array becomes one element, return its value.
2. Otherwise, if $m > n$, rotate the array 90 degrees.
3. Thus we assume $m \leq n$.
4. Let A_{left} be the solution for the left half.
5. Let A_{right} be the solution for the right half.
6. Let A_{column} be the solution for the column-centered problem.
7. Let the solution be the maximum of those three.

Here the column-centered problem is to obtain an array portion that crosses over the central vertical line with maximum sum, and can be solved in the following way.

$$A_{column} = \mathrm{MAX}_{k=0,l=0,i=1,j=n/2+1}^{i-1,n/2-1,m,n}\{s[i,j] - s[i,l] - s[k,j] + s[k,l]\}$$

In the above we first fix i and k, and maximize the above by changing l and j. Then the above problem is equivalent to maximizing the following for $i = 1 \cdots, m$ and $k = 0 \cdots, i-1$.

$$A_{column}[i,k] = \mathrm{MAX}_{l=0,j=n/2+1}^{n/2-1,n}\{-s[i,l] + s[k,l] + s[i,j] - s[k,j]\}$$

Let $s^*[i,j] = -s[j,i]$. Then the above problem can further be converted into

$$A_{column}[i,k] = -\mathrm{MIN}_{l=0}^{n/2-1}\{s[i,l]+s^*[l,k]\}+\mathrm{MAX}_{j=n/2+1}^{n}\{s[i,j]+s^*[j,k]\} \cdots (3)$$

The first part in the above is distance matrix multiplication of the MIN-version and the second part is of the MAX-version. Let S_1 and S_2 be matrices whose (i,j) elements are $s[i,j-1]$ and $s[i,j+n/2]$ for $i = 1 \cdots, m; j = 1 \cdots, n/2$. For an arbitrary matrix T, let T^* be that obtained by negating and transposing T. As the range of k is $[0\,..\,m-1]$ in S_1^* and S_2^*, we shift it to $[1..m]$. Then the above can be computed by multiplying S_1 and S_1^* by the MIN-version and taking the lower triangle, multiplying S_2 and S_2^* by the MAX-version and taking the lower triangle, and finally subtracting the former from the latter and taking the maximum from the resulting triangle. We call the operation of transforming a matrix into a triangle $triangulation$.

For simplicity, we apply the algorithm on a square array of size (n,n), where n is a power of 2. Then all parameters m and n appearing through recursion in Algorithm M are power of 2, where $m = n$ or $m = n/2$. We observe the algorithm splits the array vertically and then horizontally. We define the work of computing the three A_{column}'s through this recursion of depth 2 to be the work at level 0. The algorithm will split the array horizontally and then vertically through the next recursion of depth 2. We call this level 1, etc.

Now let us analyze the time for the work at level 0. We can multiply $(n,n/2)$ and $(n/2,n)$ matrices by 4 multiplications of size $(n/2,n/2)$, and there are two such multiplications in (3). Let $M(n)$ be the time for multiplying two $(n/2,n/2)$ matrices. At level 0, we obtain an A_{column} and two smaller A_{column}'s, spending $12M(n)$ comparisons. Thus we have the following recurrence for the total time $T(n)$. The following lemma is obvious.

$$T(1) = 0, T(n) = 4T(n/2) + 12M(n)$$

Lemma 3. *Let c be an arbitrary constant such that $c > 0$. Suppose $M(n)$ satisfies the condition $M(n) \geq (4+c)M(n/2)$. Then the above $T(n)$ satisfies $T(n) \leq 12(1 + 4/c)M(n)$.*

Clearly the complexity of $O(n^3)$ for $M(n)$ satisfies the condition of the lemma with some constant $c > 0$. Thus the maximum subarray problem can be solved in $O(n^3)$ time. Since we take the maximum of several matrices component-wise in our algorithm, we need an extra term of $O(n^2)$ in the recurrence to count the number of operations. This term can be absorbed by slightly increasing 12, the coefficient of $M(n)$.

Suppose n is not given by a power of 2. By embedding the array a in an array of size (n',n') such that n' is the next power of 2 and the gap is filled with 0, we can solve the original problem in the complexity of the same order. Similar considerations can be made on K in the following.

Now we describe the K-maximum subarray problem. When the recursion hits a (K, K) array, we select K largest sums from possible K^4 subarrays. This can easily be done by changing the top-left and bottom-right co-ordinates on the prefix sum array. Let us call this algorithm Algorithm A. Suppose K is a power of 2. If not, we can choose the next power of 2 for K. First we change line 1 in Algorithm M as follow:

1. If the array becomes $K \times K$, return the solution by Algorithm A.

Next we describe how to compute A_{column} at each recursion. We first define $\mathbf{a} - \mathbf{b}$ for two K-tuples, **a** and **b**, to be the K values that are made by subtracting elements of **b** from those of **a** component-wise. To compute distance matrix multiplication by $S_2S_2^* - S_1S_1^*$ in (4), we use the K-matrix multiplication of MAX and MIN version. To compute the subtraction, we follow the above operation of $\mathbf{a}-\mathbf{b}$ component-wise. As we assume $K \leq n$, this complexity $O(Kn^2)$ of triangulation and subtraction is absorbed in the main complexity. The initial condition for T becomes $T(K) = O(K^4)$. As there are $n/K \times n/K$ subarrays at the bottom of recursion, the total time spent by Algorithm A is $O((n/K)^2K^4)$ = $O(n^2K^2)$. If we use the $O(n^3)$ time algorithm for K-matrix multiplication in Algorithm M, the total time before hitting the bottom of recursion is $O(n^3)$. Thus the total time is $O(n^3 + n^2K^2)$. This time complexity is $O(n^3)$ when $K \leq \sqrt{n}$. The K maximum sums can be sorted with additional $O(K \log K)$ time.

6 Concluding Remarks

In the previous section, we improved the complexity from $O(n^3 \log K)$ to $O(n^3)$ for small K. If we use a sub-cubic algorithm for DMM with time complexity $O(n^3\sqrt{\frac{\log\log n}{\log n}})$ in [14], we can achieve a sub-cubic complexity for the the two-dimensional case for even smaller $K \leq O(\sqrt{\frac{\log n}{\log\log n}})$, using the same frame work of divide-and-conquer and K-tuples. Recent developments for DMM [16, 19] can also be incorporated. Details are omitted here.

If we find K-maximum subarray in a graphic image, those will heavily overlap. That is, we will find many array portions that only slightly differ in co-ordinates. If we are only interested in strictly disjoint portions, one way to solve this problem is the following greedy method. When we find the maximum sum using Algorithm 2, we replace the value of each cell comprising the maximum sum with $-\infty$, and repeat this algorithm. By repeating this process, we can find the second maximum sum, the third, etc. For a one-dimensional array, as each run takes $O(n)$ time, we can find the K-maximum subarray in $O(Kn)$ time. This is however solved in $O(n)$ time [13]. We can extend the $O(Kn)$ time algorithm to two dimensions with $O(Kn^3)$ time. It remains to be seen if we can extend the $O(n)$ time algorithm to two dimensions with $O(n^3)$ time.

The sum of those maximum subarrays by this greedy method may not be the maximum of the total sum of K disjoint subarrays. This problem of minimizing the total sum

of K disjoint subarrays has been solved in linear time for the one-dimensional case in [8]. To the authors' knowledge, the two-dimensional case has not been solved.

References

1. Bae, S.E., Takaoka, T.: Parallel approaches to the maximum subarray problem. Japan-Korea Workshop on Al. and Comp. (2003) 94–104
2. Bae, S.E., Takaoka, T.: Algorithms for the problem of K maximum sums and a VLSI algorithm for the K maximum subarrays problem. ISPAN 2004 (2004) 247–253
3. Bengtsson, F., Chen, J.: Efficient algorithms for the k maximum sums. ISAAC 2004 LNCS, Vol. 3341 Springer (2004) 137–148
4. Bentley, J.: Programming pearls: algorithm design techniques. Commun. ACM, Vol. 27(9) (1984) 865–873
5. Bentley, J.: Programming pearls: perspective on performance. Commun. ACM, Vol. 27(11) (1984) 1087–1092
6. Brodal, G.S.: Finger search trees with constant insertion time. SODA 1998 (1998) 540–549
7. Brown, M.R., Tarjan, R.E.: The design and analysis of a data structure for representing sorted lists. SIAM Jour. on Comp., Vol. 9(3) (1980) 594–614
8. Csürös, M.: Algorithms for finding maxima-scoring segment sets. WABI 2004 LNCS, Vol. 3240 Springer (2004) 62–73
9. Dietz, P.F., Raman, R.: A constant update time finger search tree. Inf. Process. Lett., Vol. 52(3) (1994) 147–154
10. Miller, R., Boxer, L.: Algorithms Sequential & Parallel- A Unified Approach. Prentice Hall, (2000)
11. Perumalla, K., Deo, N.: Parallel algorithms for maximum subsequence and maximum subarray. Parallel Process. Lett., Vol. 5(3) (1995) 367–373
12. Qui, K., Akl, S.G.: Parallel maximum sum algorithms on intercommenction networks. Queen's Uni. Dept. of Com. and Info. Sci. Technical Report 99-431 (1999)
13. Ruzzo, W.L., Tompa, M.: A linear time algorithm for finding all maximal scoring subsequences. Intelligent Sys. in Molecular Biology (1999) 234–241
14. Takaoka, T.: A new upper bound on the complexity of the all pairs shortest paths problem. Inf. Process. Lett., Vol. 43(4) (1992) 195–199
15. Takaoka, T.: Efficient algorithms for the maximum subarray problem by distance matrix multiplication. Elec. Notes in Theoretical Computer Sci., Vol. 61 Elsevier (2002)
16. Takaoka, T.: A faster algorithm for the all-pairs shortest path problem and its application. COCOON 2004, LNCS, Vol. 4106 Springer (2004) 278–289
17. Tamaki, H., Tokuyama, T: Algorithms for the maximum subarray problem based on matrix multiplication. SODA 1998 (1998) 446–452
18. Wen, Z.: Fast parallel algorithms for the maximum sum problem. Parallel Computing, Vol. 21(3) (1995) 461–466
19. Zwick, U.: A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. ISAAC 2004, LNCS, Vol. 3341 Springer (2004) 921–932

Server Allocation Algorithms for Tiered Systems*

Kamalika Chaudhuri[1], Anshul Kothari[2], Rudi Pendavingh[3], Ram Swaminathan[4], Robert Tarjan[4], and Yunhong Zhou[4,**]

[1] Computer Science Division, University of California, Berkeley, CA 94720
[2] Computer Science Depart, University of California, Santa Barbara, CA 93106
[3] TU Eindhoven, Depart. of Math. and CS., Eindhoven, The Netherlands
[4] HP Labs, 1501 Page Mill Rd, Palo Alto, CA 94304
yunhong.zhou@hp.com

Abstract. Many web-based systems have a tiered application architecture, in which a request needs to transverse all the tiers before finishing its processing. One of the most important QoS metrics for these applications is the *expected response time* for the user. Since the expected response time in any tier depends upon the number of servers allocated to this tier, and a request's total response time is the sum of the response times at all the tiers, many different configurations (number of servers allocated to each tier) can satisfy the expected response time requirement. Naturally, one would like to find the configuration to minimize the total system cost while satisfying the total response time requirement. This is modeled as a non-linear optimization problem using an open-queuing network model of response time, which we call the *server allocation problem for tiered systems* (SAPTS).

In this paper we study the computational complexity of SAPTS and design efficient algorithms to solve it. For a variable number of tiers, we show that the decision problem of SAPTS is NP-complete. Then we design a simple two-approximation algorithm and a fully polynomial time approximation scheme (FPTAS). If the number of tiers is a constant, we show that SAPTS is polynomial-time solvable. Furthermore, we design a fast polynomial-time exact algorithm to solve for the important two-tier case. Most of our results extend to the general case where each tier has an arbitrary response time function.

1 Introduction

The last few years have seen tremendous growth in the area of web based applications such as electronic commerce, web service and search engines. As these applications are user oriented, their main objective is to keep their users satisfied by meeting certain quality of service requirements. One of the most important quality of service parameters for these applications is *expected response time*, which is the total time it takes to process a user's request.

* The work of Chaudhuri and Kothari were done while they were interns at HP Labs.
** Corresponding author

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 632–643, 2005.

Many applications, especially web applications, have a tiered application architecture, in which a user's request is processed by multiple (levels of) servers. For example, a typical web-service system consists of three tiers: web servers, application servers and database servers. Within each tier, multiple machines can be provisioned to share the incoming workload, which consists of a series of different types of web requests. At each of the tiers, a user's request is going to suffer *queuing delay* and *processing delay*, and the expected response time bounds the total delay suffered by a user.

Although three tiers are typical for a web-application architecture, it is possible for an application to have more tiers. Consider a typical search engine such as Google [2]. The application server tier is actually very complex, and it consists of multiple sub-tiers, doing things such as crawling, parsing, indexing, ranking, and searching. If the service demand for one tier is very large, a few load balancing servers are placed in front of the tier to divide the load equally into multiple sub-workloads, and together these load balancing servers act as one tier.

Our question is to allocate an adequate number of servers to each tier in order to meet a certain service level requirement. Applications that allow different number of machines at each tier are called *horizontally scalable*. We deal with such a horizontally scalable system, in which the service level requirement is expressed in terms of the average response time of the system. At each tier, the queuing delay suffered by a request is a function of the number of servers in that tier. One can reduce the total delay by increasing the number of servers; but this also increases the total infrastructure cost. Thus, there is a tradeoff between the infrastructure cost and the expected response time. This necessitates a tool, which given the application's workload and expected response time, finds a server allocation (how many servers in each tier) with minimum cost.

Since the system response time is the sum of response times of each of the tiers, the total response time requirement can be met by different configurations (numbers of machines in each tier). The question, then, is to find a minimum number of machines in total to meet the average response time requirement. Realizing that machines at different tiers might be different and incur different costs, the more general optimization problem is to minimize the total weighted sum of servers, with the weights reflecting the "total cost of ownership" of servers in different tiers.

In this paper, we model this problem as a non-linear integer optimization problem, and call it the *server allocation problem for tiered systems* (SAPTS). For the case where the system has a variable number of tiers, we first show that the decision version of SAPTS is NP-complete. We then present a simple two-approximation algorithm based on Lagrangian relaxation. Next, we present a pseudo-polynomial-time algorithm and a fully polynomial time approximation scheme (FPTAS). If the number of tiers is a constant, we show that the problem is polynomial-time solvable using a variant of Lenstra's algorithm [11]. For the important special case of two tiers, we design a fast polynomial-time exact algorithm. We also generalize our results to arbitrary response time functions.

1.1 Related Work

The problem we consider in this paper is a continuation of TAO (web transaction and optimization), a HP project focusing on developing metrics, models, and infrastructures to effectively manage the performance of web applications. See Garg et al. [5] for details on system performance modeling. Zhang et al. [13] have modeled the server allocation problem as a non-linear integer optimization problem and proposed a search heuristic to solve it optimally. The search heuristic only considers optimal solutions and has exponential running time.

Our work is related to both capacity planning and resource allocation. Menasce and Almeida [12] present general techniques of capacity planning for web applications. For resource allocation, the ability to dynamically allocating computing resources in a shared resource environment is essential of utility computing. Appleby et al. [1] describe various aspects of IBM's Océano project which is centered on SLA management for utility computing. A key problem with utility computing is that the user needs are translated into a logical configuration, and physical resources are then assigned to the logical configuration to satisfy the resource requirements of the application. The problem addressed in this paper is precisely to determine the optimal resource requirements of an application under a workload in such a way that application SLAs are satisfied.

Zhu and Singhal [14] addressed the issue of allocating resources (machines) in a tree-like topology of a data center, considering performance constraints such as link bandwidth and switch capacity while minimizing communication traffic among the assigned servers. They propose a mathematical optimization model with binary variables for optimally configuring the topology. Our work differs from [14] in several important ways. First, we consider only the average response time performance measure. Second, our topology is a tiered structure compared to an arbitrary tree topology. These simplifications allow us to devise efficient algorithms to solve the server allocation problem.

Operations Researchers have studied resource allocation as an optimization problem, and variations of knapsack problems have been proposed to solve this problem. SAPTS is actually the *dual* of the *non-linear knapsack problem*. The special case where $h_i = 1$ for all i's (servers cross tiers have identical costs) is called the *simple allocation problem* and solved by Frederickson and Johnson [4] in time $O(k \log(p/k))$, where k is the number of tiers and p is the upper bound for the number of total machines. For the general case where h_i's are arbitrary integers, Hochbaum [8] gives a FPTAS for it. However, both [4, 8] rely on the crucial assumption that the response time functions are *convex*, in order to convert the original problem into an equivalent selection problem or knapsack problem. Our method converts the original problem into a multi-choice knapsack problem, and so no convexity properties are needed for the response time functions.

The rest of the paper is organized as follows. We describe the response time model in Section 2.1 and formulate the server allocation problem as a mathematical optimization problem in Section 2.2. In Section 3.1, we show that SAPTS is NP-complete for variable number of tiers. We give a two-approximation algorithm in Section 3.2, a pseudo-polynomial-time algorithm as well a FPTAS in

Section 3.3. We give a simple polynomial-time algorithm for the important two-tier case in Section 4.1 and show that the problem is polynomial-time solvable for constant number of tiers in Section 4.2. We discuss arbitrary response time functions in Section 5 and conclude in Section 6.

2 Problem Formalization

We first describe the response time model, and then state SAPTS precisely using this model.

2.1 The Response Time Model

One of the nice properties of tiered systems is that the delay suffered by a request in a tier only depends upon the number of servers in that tier and is not affected by the number of servers in any other tier. Therefore, one can compute a request's response time by computing the delays at individual tiers and summing them to obtain the total response time.

To simplify our modeling effort for each tier, we assume that all servers in the same tier are identical, and that the workload is shared approximately equally among all the servers in the same tier. If the request arrival rate is λ_i for the i-th tier with N_i servers, then each server has a request arrival rate of λ_i/N_i. Each server is modeled as a processor sharing queue. The expected response time is given by the expected time in system with an M/M/1 queuing model:

$$R_i(N_i) = \frac{E[S_i]}{1 - (\frac{\lambda_i}{N_i})E[S_i]} \ , \tag{1}$$

where $E[S_i]$ is the expected processing time (or service demand) of a request on the critical resource (such as the CPU) at the i-th tier[1]. (For a reference to general queuing theory and the above formula, see Kleinrock [10].) As discussed before, the response time, $R(N)$, for a k-tier application is the sum of the delays at all the tiers. Therefore

$$R(N) \ = \ \sum_{i=1}^{k} R_i(N_i) \ = \ \sum_{i=1}^{k} \frac{E[S_i]}{1 - \frac{\lambda_i E[S_i]}{N_i}}, \tag{2}$$

where $N = (N_1, N_2, \ldots, N_k)$. We refer to N as a *configuration*.

2.2 The Server Allocation Problem for Tiered Systems

From Eq. (2), it follows that there exist many system configurations that satisfy the response time bound. Among these feasible allocations, one would like to find the one with the minimum cost. This is formulated as an optimization problem:

[1] $E[S_i]$ can be estimated from the measured utilization rate, u_i, of the critical resource as follows: $E[S_i] = u_i/(\lambda_i/N_i)$.

$$\min_{N_i} \sum_{i=1}^{k} h_i N_i \tag{3}$$

$$\text{s.t. } R(N) = \sum_{i=1}^{k} \frac{E[S_i]}{1 - \frac{\lambda_i E[S_i]}{N_i}} \leq T_0; \tag{4}$$

$$N_i \text{ integer with } N_i > \lambda_i E[S_i], \text{ for } i = 1, \ldots, k,$$

where T_0 is the required response time and the weights h_i (all assumed to be strictly positive) are the costs of servers in different tiers. Because

$$\frac{E[S_i]}{1 - \frac{\lambda_i E[S_i]}{N_i}} = \frac{N_i E[S_i]}{N_i - \lambda_i E[S_i]} = E[S_i] + \frac{\lambda_i E[S_i]^2}{N_i - \lambda_i E[S_i]}, \quad \text{for all } i,$$

the non-linear constraint (4) can be further simplified as follows:

$$\sum_{i=1}^{k} \frac{\lambda_i E[S_i]^2}{N_i - \lambda_i E[S_i]} \leq T_0 - \sum_{i=1}^{k} E[S_i].$$

Let $a_i = \lambda_i E[S_i]^2$, $b_i = \lambda_i E[S_i]$ and $T = T_0 - \sum_{i=1}^{k} E[S_i]$, then the response time constraint becomes:

$$\sum_{i=1}^{k} \frac{a_i}{N_i - b_i} \leq T. \tag{5}$$

Given the optimization formulation of SAPTS, one can define the corresponding *decision* problem (dSAPTS) as follows: for a given cost p, does there exist an allocation $N = (N_1, \ldots, N_k)$ such that,

$$\sum_{i=1}^{k} h_i N_i \leq p$$

$$\sum_{i=1}^{k} \frac{a_i}{N_i - b_i} \leq T$$

$$N_i \text{ integer with } N_i > b_i, \text{ for } i = 1, \ldots, k.$$

For simplicity, we assume that h_i's are positive integers for all i. However, a_i, b_i are not necessarily integers. It is easy to see that if one can solve the decision problem in polynomial time then the optimization problem can also be solved in polynomial time by doing a binary search on possible values of p. Thus from the computational complexity point of view, SAPTS is equivalent to dSAPTS.

3 Variable Number of Tiers

In this section, we present both approximate and exact algorithms for SAPTS when the number of tiers is a variable. We first show in Section 3.1 that dSAPTS is NP-complete if the number of tiers is a variable. Then in Section 3.2, we give a simple two-approximation based on Lagrangian relaxation. In Section 3.3, we first give a pseudo-polynomial time algorithm by converting SAPTS into a multi-choice knapsack problem (MCKP), then use standard scaling techniques to obtain a fully polynomial time approximation scheme.

3.1 NP-Completeness

Our first result shows that SAPTS is "hard" to solve optimally in polynomial time when the number of tiers is a variable.

Theorem 1. *The decision problem of SAPTS is NP-complete, assuming that the number of tiers is a variable.*

3.2 A Two Approximation Algorithm

In the previous section, we have shown that the server allocation problem is *NP-complete* for arbitrary k. Given the hardness of computing the optimal solution one would like to know if it is possible to compute a good approximate solution efficiently. It turns out that a fully polynomial time approximation scheme (FPTAS) exists for our problem. Before presenting the relatively complex approximation scheme, we first give a simple approximation algorithm which runs in linear time. Our simple approximation algorithm guarantees a worst-case performance factor of two, and it will be used for the construction of our FPTAS in Section 3.3.

By relaxing the constraint that N_i's have to be integers, we can use the Lagrangian multiplier method to compute a closed form solution for the relaxed optimization problem. By rounding up the solution for the relaxed optimization problem, we get an approximate solution with cost less than twice the minimum cost. Consider the Lagrangian function where λ is the Lagrangian multiplier:

$$L(N_1, N_2, \cdots, N_k, \lambda) = \sum_{i=1}^{k} h_i N_i + \lambda \left(\sum_{i=1}^{k} \frac{a_i}{N_i - b_i} - T \right).$$

The optimal fractional value N_i^f has the following closed form (details of the calculation omitted):

$$N_i^f = b_i + \sqrt{\frac{a_i}{h_i}} \cdot \frac{\sum_{j=1}^{k} \sqrt{h_j a_j}}{T}, \quad i = 1, \ldots, k.$$

The optimal fractional solution N^f can be converted into a feasible two-approximation directly. Let N^r be the integer solution got by rounding up N^f. Specifically, $N_i^r = \lceil N_i^f \rceil$, for all i. Because $N_i^r \geq N_i^f$ for all i, $\sum_{i=1}^{k} a_i/(N_i^r - b_i) \leq \sum_{i=1}^{k} a_i/(N_i^f - b_i) = T$. Therefore N^r satisfies the total response time constraint and N^r is a feasible solution to SAPTS. Next, we show that the cost(N^r) is within twice of cost(N^*), where N^* is an optimal integral solution. Because $N_i^r = \lceil N_i^f \rceil$, it is easy to see that $1 \leq N_i^r < N_i^f + 1$, $\forall\, i$. Therefore

$$\text{cost}(N^r) = \sum_{i=1}^{k} h_i N_i^r < \sum_{i=1}^{k} h_i (N_i^f + 1) = \text{cost}(N^f) + \sum_{i=1}^{k} h_i. \tag{6}$$

Since N^f is the optimal fractional solution, its cost is no more than the cost of the optimal integral solution N^*, i.e., cost$(N^f) \leq$ cost(N^*). Also, since any optimal integral solution should at least contain one server in each of the tiers, we have $\sum_i h_i \leq$ cost(N^*). Eq. (6) together with the above two inequalities implies that cost$(N^r) \leq 2$cost(N^*). In summary, we have the following theorem:

Theorem 2. *For SAPTS with k tiers where k is an arbitrary integer variable, we can compute a two-approximation in time $O(k)$.*

It is easy to construct pathological examples where the performance ratio of the above algorithm versus the optimal is arbitrarily close to two, and so our analysis is tight. However, in practice where T is small, we expect the algorithm to work very well, with its performance ratio close to 1. This is because Eq. (6) together with the fact $\text{cost}(N^f) \leq \text{cost}(N^*)$ leads to

$$\frac{\text{cost}(N^r)}{\text{cost}(N^*)} \leq \frac{\text{cost}(N^r)}{\text{cost}(N^f)} \leq 1 + \frac{\sum_{i=1}^{k} h_i}{\text{cost}(N^f)}.$$

When T approaches 0, $\text{cost}(N^f)$ approaches ∞, i.e., the above algorithm has close-to-optimum cost when T is very small.

3.3 Pseudo-Polynomial-Time Algorithm and FPTAS

In this section, we present a pseudo-polynomial time algorithm as well as a fully polynomial time approximation scheme. We start by transforming SAPTS into an equivalent multiple-choice knapsack problem. By bounding the size of each class of items, first we are able to give a pseudo-polynomial-time algorithm for our problem. This solution in turn is adapted to design the FPTAS using standard scaling techniques.

The multi-choice knapsack problem is a generalization of the ordinary knapsack problem, where there are m sets of items $S_1, \ldots, S_m$ with $|S_i| = n_i$ for all i and $\sum_{i=1}^{m} n_i = n$. Each item $j \in S_i$ consists of a weight w_{ij} and a profit p_{ij}, and we are given a knapsack with capacity W. The objective is to pick exactly one item from each set, such that the profit sum is maximized and the weight sum is bounded by the knapsack capacity W. It is well-known that MCKP is NP-complete. An approximation algorithm with performance guarantee 5/4 is given by Gens and Levner [6]. A pseudo-polynomial time algorithm using dynamic programming is easy to design. The first FPTAS for MCKP has been given by Chandra, Hirschberg and Wong [3]. An improved FPTAS is given by Kellerer et al. [9] using standard scaling with time complexity $O(nm \cdot 1/\epsilon)$. See [9] also for an extensive treatment of knapsack problems, including MCKP.

Given a SAPTS instance with k tiers, the multi-choice knapsack problem consists of k sets of items $S_1, \ldots, S_k$. For an item $j \in S_i$:

$$\text{weight}(j) = \frac{a_i}{j - b_i}, \quad \text{cost}(j) = j \cdot h_i.$$

For the optimization problem of SAPTS, we don't have any cost constraint, thus any $j > b_i$ is a feasible item in S_i. In order to bound the size of S_i, we bound the number of servers needed at each tier. Consider the response time constraint:

$$\frac{a_i}{N_i - b_i} \leq \sum_{i=1}^{k} \frac{a_i}{N_i - b_i} \leq T \quad \Rightarrow \quad N_i \geq b_i + \frac{a_i}{T} \equiv n_i^l.$$

Thus, n_i^l is a lower bound on the number of servers needed at the i-th tier.

Next, let us consider the two-approximation solution N^r. Let $C^r = \text{cost}(N^r)$. Since C^r is an upper bound on total cost of the optimal, we have:

$$h_i N_i + \sum_{j|j\neq i} h_j n_j^l \leq \sum_{j=1}^{k} h_j N_j \leq C^r \quad \Rightarrow \quad N_i \leq \frac{C^r - \sum_{j|j\neq i} h_j n_j^l}{h_i} \equiv n_i^u.$$

Thus, n_i^u is an upper bound on the number of servers needed at i-th tier.

Given n_i^l and n_i^u, we only need to consider item $j \in S_i$ satisfying $\lceil n_i^l \rceil \leq j \leq \lfloor n_i^u \rfloor$. Without loss of generality, we assume that n_i^l and n_i^u are both integers. Otherwise we can always replace n_i^l by $\lceil n_i^l \rceil$ and replace n_i^u by $\lfloor n_i^u \rfloor$. Therefore now set S_i has only $n_i^u - n_i^l + 1$ elements. Next we describe a pseudo-polynomial time algorithm to solve MCKP.

The multi-choice knapsack problem can be solved, in pseudo polynomial time, by building a *dynamic programming* (DP) table. There are two ways to build the dynamic programming table; either using the cost or using the weight. As the weights of items in our case are not integers, and the costs are integers, we build it using the cost. Let $F(i, c)$ denote the minimum weight of items selected from the first i item sets with total cost bounded by c. Following is the recursion function to build the DP table F:

$$F(i, c) = \min_{j \in S_i} \{F(i-1, c - \text{cost}(j)) + \text{weight}(j)\}$$

From Theorem 2, C^r (the cost of N^r) is within twice of cost(N^*), the optimum cost. We restrict the size of the cost parameter (number of columns) of the table to C^r. Thus, the total time taken to build the table is $O(C^r \cdot \sum_{i=1}^{k}(n_i^u - n_i^l + 1))$. Once the table has been built, the optimal solution can be found by going through the last row and choosing the minimum cost c, such that $F(k, c)$ is bounded by at most T. Thus, we have the following result:

Theorem 3. *SAPTS is pseudo-polynomial-time solvable. The optimal solution can be computed in time* $O(C^r \cdot \sum_{i=1}^{k}(n_i^u - n_i^l + 1)) = O(cost(N^*) \cdot \sum_{i=1}^{k}(n_i^u - n_i^l + 1))$.

The pseudo polynomial algorithm given above can be converted into a fully polynomial time approximation scheme using cost scaling. The following theorem is our main result in this section:

Theorem 4. *For SAPTS with k tiers, we can compute a $(1+\varepsilon)$-approximation with time $O(k^3 \cdot 1/\varepsilon^2)$ and space $O(k^2 \cdot 1/\varepsilon)$.*

4 Constant Number of Tiers

We showed in Section 3.1 that SAPTS is NP-Hard for arbitrary number of tiers. However, real world tiered systems are usually composed of only a small number of tiers. For example, a typical ecommerce application has only three tiers: a web server tier, an application server tier, and a database server tier. So it is natural to ask if one could do better if the number of tiers is small.

4.1 A Polynomial Time Algorithm for Two Tiers

As before, in order to solve SAPTS optimally, it suffices to solve the decision problem dSAPTS optimally. If the system has only two tiers, thus it suffices to given an algorithm to determine, in polynomial time, whether the set

$$\{(x_1, x_2) \in \mathbb{R}^2 \mid h_1 x_1 + h_2 x_2 \leq p, \frac{a_1}{x_1 - b_1} + \frac{a_2}{x_2 - b_2} \leq T\}$$

contains an integer vector, given positive integers $h_1,\ h_2,\ p$ and nonnegative reals $a_1,\ b_1,\ a_2,\ b_2,\ T$.

Let $f : x \mapsto \frac{a_1}{x_1 - b_1} + \frac{a_2}{x_2 - b_2}$. Here's the algorithm.

1 Compute the positive integers h'_1, h'_2 so that $|h_1 h'_2 - h_2 h'_1| = 1$;
2 **repeat**
3 Determine the point x^* on $L := \{x \in \mathbb{R}^2 \mid hx = p\}$ minimizing $f(x)$;
4 If $f(x^*) > T$, **return** "*no*";
5 Let x', x'' be the integer points on either side of x^* on L;
6 If $f(x') \leq T$ or $f(x'') \leq T$ **return** "*yes*";
7 Put $h' \leftarrow h' - \lfloor < h, h' > \rfloor h$; if $h' \leq 0$, put $h' \leftarrow -h'$;
8 Put $p' \leftarrow \min\{h'x', h'x''\}$;
9 If $p' < \min\{h'x \mid hx \leq p, f(x) \leq T, x > b\}$, then **return** "*no*";
10 Swap h and h', put $p \leftarrow p'$;
11 **end repeat**

Here $x \in \mathbb{R}^2$ is a column vector, h is the rowvector (h_1, h_2), h' is the rowvector (h'_1, h'_2), and $< h, h' >$ is their inner product.

Theorem 5. *The above algorithm decides, given integers h_1, h_2, p and reals a_1, b_1, a_2, b_2, T, whether*

$$\{(x_1, x_2) \in \mathbb{R}^2 \mid h_1 x_1 + h_2 x_2 \leq p, \frac{a_1}{x_1 - b_1} + \frac{a_2}{x_2 - b_2} \leq T, x_1 \geq b_1, x_2 \geq b_2\}$$

contains an integer vector, in $O(\log(h_1) + \log(h_2))$ arithmetic operations on real numbers and integers.

4.2 Lenstra's Algorithm and Its Application to SAPTS

It turns out that it is indeed possible to solve SAPTS in polynomial time when the number of tiers is a constant. This can be done by a variant of Lenstra's algorithm for solving integer linear programs with constant number of variables. The details of the following theorem can be found in the book by Grőtschel, László Lovász and Schrijver [7]:

Theorem 6. *Suppose we have an k-dimensional convex body P described by a separation oracle. If k is a constant, it is possible to determine in polynomial time if there is an integer point inside P.*

Given the polynomial-time equivalence of the optimization problem and the decision problem, we show how to solve the decision version of SAPTS in this

subsection. The corresponding body P in SAPTS is formed by the surface $\sum_{i=1}^{k} a_i/(N_i - b_i) \leq T$ and the hyperplane $\sum_{i=1}^{k} h_i N_i \leq p$. If there is an integer point inside P, there exists a solution for the decision problem of SAPTS with cost $\leq p$.

It is easy to see that the body P in our problem is convex. Designing a separation oracle for our convex body P is easy; for a point that violates a linear constraint, we take the same constraint as the separating hyperplane. For a point o that violates a non-linear constraint, we draw a line between o and some arbitrary point q inside P. Let r be the point where this line intersects P; the tangent at r to the non-linear constraint curve will be our separating hyperplane. The following is therefore a corollary of Theorem 6.

Corollary 1. *SAPTS with a constant number of tiers can be solved in polynomial time.*

5 Arbitrary Response Time Functions

As evidenced in [5], the response time model presented in Section 2.1 is a good approximation of realistic system setups. However, different workload characteristics and hardware configurations will result in different response time models. Fortunately, our algorithms for SAPTS actually don't depend on the specific form of the response time functions. Therefore in this section we revisit SAPTS with general response time functions, which we call gSAPTS.

As before, we assume that all servers in the same tier are identical and have the same response time. If the request arrival rate is λ_i for the i-th tier with N_i servers, then its response function becomes $R_i(N_i, \lambda_i) = f_i(N_i)$ for $i = 1, \ldots, k$. Let $N = (N_1, N_2, \ldots, N_k)$ and $R(N)$ be the total response time for the tiered system. As discussed before, $R(N)$ is the sum of the delays at all the tiers, i.e., $R(N) = \sum_{i=1}^{k} R_i(N_i, \lambda_i) = \sum_{i=1}^{k} f_i(N_i)$. For a given T_0, there exist multiple server allocations that satisfy the response time bound. Among these feasible configurations, one would like to find the one with the minimum cost. This is formulated as the following optimization problem, which we call *General Server Allocation Problem for Tiered Systems* (gSAPTS):

$$\min_{N_i} \sum_{i=1}^{k} h_i N_i \tag{7}$$

$$\text{s.t.} \sum_{i=1}^{k} f_i(N_i) \leq T_0; \tag{8}$$

$$N_i \text{ integers with } N_i \geq n_i^l, \text{ for } i = 1, \ldots, k,$$

where T_0 is the required response time and the weights h_i (all assumed to be strictly positive) are the costs of servers in different tiers. Function f_i takes positive integer domains with $f_i(N_i) < \infty$ for $N_i \geq n_i^l$ and $f_i(N_i) = \infty$ otherwise. Here n_i^l is a positive integer, with the trivial case $n_i^l = 1$, for $i = 1, \ldots, k$.

gSAPTS can be converted into MCKP similar as the case for SAPTS. Given an instance of gSAPTS, the MCKP consists of k sets of items $S_1, S_2, \ldots, S_k$. For an item $j \in S_i$, its weight and cost are:

$$\text{weight}(j) \;=\; f_i(j), \quad \text{cost}(j) \;=\; j \cdot h_i.$$

Here $j \in S_i$ is lower bounded by n_i^l. The challenge is to come up with an upper bound for the number of servers in the i-th tier, for all index i.

In the following we use cost doubling to determine the optimal cost value. Initially we set $C_0 = 2\sum_{i=1}^{k} h_i n_i^l$ and $n_i^u = \lfloor C_0/h_i \rfloor$, for $i = 1, \ldots, k$. Now each set S_i has exactly $n_i^u - n_i^l + 1$ elements and we can run the DP-by-cost procedure as described in Section 3.3 to compute the optimum solution for this MCKP instance. If the procedure returns an optimal feasible solution, then we return that solution and are done. Otherwise, the procedure will tell us that there is no feasible solution for this MCKP instance. Now we double C_0 by setting $C_0 := 2C_0$ and repeat the above procedure. And the program ends at the first time when the DP program returns an optimal feasible solution. Formally the procedure is described below:

1 Let $C_0 = 2\sum_{i=1}^{k} h_i n_i^l$.
2 Set $n_i^u = \lfloor C_0/h_i \rfloor$, for $i = 1, \ldots, k$.
3 Construct sets $S_1, \ldots, S_k$ where
4 $\quad S_i \;=\; \{j \;\mid\; \text{weight}(j) = f_i(j),\ \text{cost}(j) = j \cdot h_i,\ n_i^l \le j \le n_i^u\}$.
5 Solve the multi-choice knapsack problem using DP-by-cost.
6 If DP-by-cost returns an optimal feasible solution, return the solution
7 $\quad$ and stop; $\quad$ else set $C_0 = 2C_0$, go back to step 2.

It is easy to verify the correctness of the algorithm. In terms of its running time, it is dominated by the last run of the algorithm, where the value of C_0 is in the range of $[\text{cost}(N^*), 2\text{cost}(N^*))$, where N^* is the minimum cost solution for gSAPTS. We summary it as the following theorem:

Theorem 7. *For arbitrary response time functions, we can solve gSAPTS in time* $O(cost(N^*) \cdot \sum_{i=1}^{k}(n_i^u - n_i^l + 1))$ *where* $n_i^u = \lfloor 2cost(N^*)/h_i \rfloor$ *for all* i*, and* N^* *is the minimum cost integer solution.*

For servers in tier i, it really doesn't make economical sense to add more servers if it results in prolonged response time. So it is natural to assume that the response time function f_i is monotone decreasing with N_i. By assuming f_i's are monotone decreasing for all i, we can convert the above algorithm into an approximation scheme using cost scaling as described in the proof of Theorem 4. The following theorem summarizes this observation:

Theorem 8. *If response time functions are monotone decreasing for all the tiers, the above algorithm can be converted into an approximation scheme using cost scaling. Specifically, we can compute an* $(1+\epsilon)$*-approximation in time* $O(k^3 \cdot 1/\epsilon^2 \cdot \log cost(N^*))$ *and space* $O(k^2 \cdot 1/\epsilon)$.

6 Concluding Remarks

In this paper, we studied the server allocation problem for tiered systems, as arisen out of resource allocation in a utility computing environment. We showed that SAPTS is NP-Hard if the number of tiers is a variable and polynomial-time solvable if the number of tiers is a constant. We presented both approximate and exact algorithms to solve this problem. In particular, we presented a fast polynomial-time algorithm to solve the important two-tier special case.

We leave with a few open problems: (i) Extend these algorithms to a more general application architecture such as a DAG structure; (ii) Design simple and efficient polynomial-time exact algorithms for the important 3-tier special case; and (iii) Handle the case where there are (small) generations of machines described by a (speed, cost) matrix.

Acknowledgements

We would like to thank the following people from HP Labs for their very helpful feedbacks and suggestions: Alex Zhang, Terence Kelly, Cipriano (Pano) Santos and Sharad Singhal.

References

1. K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, and M. Kalantar. Océano – SLA-based management of a computing utility. In *Proc. 7th IFIP/IEEE Intl. Symp. on Integrated Network Management*, May 2001.
2. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30:107–117, 1998.
3. A. Chandra, D. Hirschberg, and C. Wong. Approximate algorithms for some generalized knapsack problems. *Theoretical Computer Science*, 3:293–304, 1976.
4. G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in x+y and matrices with sorted columns. *Journal of Computer and System Sciences*, 24(2):197–208, 1982.
5. P. K. Garg, M. Hao, C. Santos, H.-K. Tang, and A. Zhang. Web transaction analysis and optimization (TAO). In *Proceedings of the 3rd Workshop on Software and Performance*, pages 286–293, 2002.
6. G. Gens and E. Levner. Approximation algorithms for certain universal problems in scheduling theory. *Soviet J. of Computers & System Sciences*, 6:31–36, 1978.
7. M. Grőtschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1988.
8. D. S. Hochbaum. A nonlinear knapsack problem. *Operations Research Letters*, (17):103–110, 1995.
9. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.
10. L. Kleinrock. *Queueing Systems, V.II: Computer Applications*. Wiley, 1976.
11. H. W. Lenstra. Integer linear programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.
12. D. A. Menasce and V. A. Almeida. *Capacity Planning for Web Performance*.
13. A. Zhang et al. Optimal server resource allocation using an open queueing network model of response time. Technical Report HPL-2002-301, HP Labs, 2002.
14. X. Zhu and S. Singhal. Optimal resource assignment in internet data centers. In *Proc. 9th MASCOTS*, pages 61–69, Cincinnati, OH, August 15-18 2001.

An Improved Approximation Algorithm for Uncapacitated Facility Location Problem with Penalties*

Guang Xu and Jinhui Xu

Department of Computer Science and Engineering
Stete University of New York at Buffalo
Buffalo, New York 14260, USA

Abstract. In this paper, we consider an interesting variant of the facility location problem called *uncapacitated facility location problem with penalties* (UFLWP for short) in which each client is either assigned to an opened facility or rejected by paying a penalty. We present a 1.8526-approximation algorithm for the UFLWP problem. Our algorithm first enhances the primal-dual method for the UFLWP problem [3] so that outliers can be recognized more efficiently, and then applies a local search heuristic to further reduce the cost for serving those non-rejected clients.

Keywords: Algorithms; Approximation Algorithms; Facility Location Problem; Outliers

1 Introduction

Facility location problem is a fundamental problem and has been extensively studied in operations research and theoretical computer science [12]. Among all its variants, the uncapacitated facility location problem has received significant amount of attention and a number of efficient approaches have been discovered in recent years [1, 2, 8, 13]. Such techniques could also be combined and extended to achieve effective approximation algorithms for other variants of facility location problem [4, 5, 7, 15].

In this paper, we study an interesting variant of the facility location problem called *uncapacitated facility location problem with penalties* (UFLWP). The uncapacitated facility location problem with penalties can be defined as follows. We are given a bipartite graph G with bipartition $(\mathcal{F},\mathcal{C})$, where $\mathcal{F}$ is the set of candidate locations for opening facilities and $\mathcal{C}$ is the set of locations (called clients) with demands for services. Each location $i \in \mathcal{F}$ is associated with a non-negative opening cost f_i, and each client $j \in \mathcal{C}$ is associated with two positive numbers d_j and p_j specifying its demands and penalty, respectively. The demands of client j could be either satisfied by assigning j to some opened facility i by paying a connection cost $d_j c_{ij}$ or rejected by paying a penalty p_j, where c_{ij}

* This research was supported in part by an IBM faculty partnership award, and an IRCAF award from SUNY Buffalo.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 644–653, 2005.

is the distance between i and j. The objective of the UFLWP problem is to determine for each client whether its demands should be rejected or satisfied so as to minimize the sum of the total penalties, connection costs and opening costs. We assume that the distance function between clients and facilities is metric, i.e., $\forall i, i' \in \mathcal{F}, \forall j, j' \in \mathcal{C}$, $c_{ij} \leq c_{i'j} + c_{ij'} + c_{i'j'}$.

Three constant approximation algorithms were proposed for the UFLWP problem [3, 9, 14]. In [3], Charikar *et. al.* achieved a 3-approximation algorithm by using a primal-dual method. Later, Jain *et. al.* gave an improved 2-approximation algorithm by applying an elegant dual fitting technique [9]. Both algorithms can be viewed as primal-dual method based algorithms. Recently, Xu and Xu showed that techniques from other categories are also capable of solving this problem and presented an LP rounding based $(2 + 2/e)$-approximation algorithm [14]. The only known hardness result of the UFLWP problem is the 1.463-inapproximability result inherited from the uncapacitated facility location problem [6]. There is a considerably large gap between the best known performance ratio and the inapproximability result. Thus it would be very interesting to know whether the performance ratio of the UFLWP problem can be further improved. In this paper, we give an affirmative answer to this question by presenting a 1.8526-approximation algorithm for the UFLWP problem.

The main difficulty of the UFLWP problem is due to the existence of penalties which disturb the metric property of the cost function between clients and facilities. Almost all existing quality-guaranteed algorithms for the facility location problems assume metric property for their cost functions. To overcome the difficulty introduced by the partial loss of metric property, we present in this paper a two-phase combinatorial algorithm. Our algorithm first uses a primal-dual method similar to the one in [3]. To avoid the large penalty on the performance ratio caused by the primal-dual method in [3], our algorithm applies the primal-dual method to a scaled instance of the problem so that outliers can be identified more accurately. The resulting solution forms an initial solution to the UFLWP problem. In the second phase, our algorithm iteratively refines the initial solution by performing a greedy local search heuristic [1, 6, 10] to open more facilities for those remaining clients so that the total facility cost and connection cost of the initial solution is reduced. Comparing with existing approaches for the UFLWP problem, our approach combines the power of the primal-dual method and local search techniques and provides a better handling of the outliers. Further, our algorithm is very simple, natural, and can be easily implemented.

2 Algorithm and Analysis

2.1 Integer Program, Linear Relaxation, and Dual

The following integer program (denoted as IP) is a natural formulation of the UFLWP problem.

$$\text{minimize} \sum_{i \in \mathcal{F}} f_i y_i + \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} d_j c_{ij} x_{ij} + \sum_{j \in \mathcal{C}} p_j z_j \tag{1}$$

$$\sum_{i\in\mathcal{F}} x_{ij} + z_j \quad \geq 1, \quad \text{for each } j \in \mathcal{C}, \tag{2}$$

$$x_{ij} \quad \leq y_i, \quad \text{for each } i \in \mathcal{F},\ j \in \mathcal{C}, \tag{3}$$

$$x_{ij} \in \{0,1\}, \text{for each } i \in \mathcal{F}, j \in \mathcal{C}, \tag{4}$$

$$y_i \in \{0,1\}, \text{for each } i \in \mathcal{F}, \tag{5}$$

$$z_j \in \{0,1\}, \text{for each } j \in \mathcal{C}. \tag{6}$$

Replacing Constraints (4), (5) and (6) in IP by Constraints $x_{ij} \geq 0$, $y_i \geq 0$, and $z_j \geq 0$ respectively, we obtain the relaxed linear program of the UFLWP problem, denoted as LP-PRIMAL. The dual (denoted as LP-DUAL) of LP-PRIMAL can be written as the following form after a simple scaling change.

$$\text{maximize} \sum_{j\in\mathcal{C}} d_j \alpha_j \tag{7}$$

$$\sum_j \beta_{ij} d_j \quad \leq f_i \quad \text{for each } i \in \mathcal{F}, \tag{8}$$

$$\alpha_j \leq \beta_{ij} + c_{ij} \text{ for each } i \in \mathcal{F}, j \in \mathcal{C}, \tag{9}$$

$$\alpha_j \quad \leq \frac{p_j}{d_j} \quad \text{for each } j \in \mathcal{C}, \tag{10}$$

$$\alpha_j, \beta_{ij} \quad \geq 0 \quad \text{for each } i \in \mathcal{F}, j \in \mathcal{C}, \tag{11}$$

2.2 Primal-Dual Method

In [3], a primal-dual method has been used to achieve a 3-approximation algorithm for the UFLWP problem. In this paper, we use a similar method to obtain an initial solution to the UFLWP problem. For clarity and completeness, below we sketch the main idea of this method.

The primal-dual method starts with a trivial feasible solution ($\alpha = 0$, $\beta = 0$) to the LP-Dual and grows all α_j, $j \in C$ at the same speed. Let τ be the time. Whenever Constraint (10) becomes tight for client j, i.e., $\tau = \frac{p_j}{d_j}$, α_j stops growing. If Constraint (9) becomes tight (at time $\tau = c_{ij}$) before Constraint (10) for some client j, then β_{ij} starts to grow at the same speed in order to maintain the feasibility of Constraint (9) until either Constraint (10) or Constraint (8) becomes tight for some $i \in F$ (i.e., $\sum_j d_j \beta_{ij} = f_i$, for some $i \in F$). After all α_j stop growing, a feasible dual solution (α, β) is obtained. An edge (i, j) is called tight if $\alpha_j = c_{ij}$. An edge (i, j) is called special if $\beta_{ij} > 0$. If $\sum_j d_j \beta_{ij} = f_i$ at time τ_i, the facility location i is called a *paid for location* and i is called a *connecting witness* for all client locations j with positive β_{ij}. If α_j stops growing at time $\tau = c_{ij}$ for a paid for location i, i is also called the connecting witness of client location j. A client location j is called a timeout vertex if α_j reaches p_j/d_j before it gets a connecting witness. Let F_t denote the set of paid for locations. Consider the subgraph T of the input bipartite graph, which contains only the special edges. Construct a graph $H = (F_t, E)$, where edge $(u, v) \in E$ if there is

a path of length at most 2 between u and v in graph T. Once H is constructed, a maximal independent set of H is computed and the corresponding facilities are opened at such locations. We say a client location j is directly connected if either a facility is opened at one of its neighbors in graph H or a facility is opened at i and edge (i, j) is a tight edge. All the remaining client locations are called indirectly connected. A timeout client location j is rejected if and only if j is indirectly connected. Let F be the facility cost, C be the connection cost, P be the penalties of rejecting clients obtained by the primal-dual method. The following lemma has been proved in [3].

Lemma 1. $3F + 3P + C \leq 3\sum_{j\in\mathcal{C}} \alpha_j$.

2.3 Scaling and Primal-Dual Method

Let us take a closer look at the primal-dual method. If we directly run the primal-dual method for the UFLWP problem, we will obtain a solution with facility cost F, rejecting penalties P, and connection cost C satisfying $3F + 3P + C \leq 3\sum_{j\in\mathcal{C}} \alpha_j$. The fact that F has a coefficient 3 indicates that there is still some room for extra facility cost in the primal dual method. Opening another set of facilities with slightly larger facility cost may enable us to reduce the connection cost and penalties and result in a solution with improved quality. Another set of facilities with slightly larger facility cost can be taken into consideration by scaling the facility cost and apply the primal dual method to the scaled instance. Therefore, we have the following algorithm.

Let $\lambda \in (1/3, 1)$ be a constant to be explicitly determined later.

1. For each facility location $i \in \mathcal{F}$, the facility cost is scaled to $f'_i = \lambda f_i$.
2. Apply the primal dual method on the scaled instance.

Let $OPT = (\bar{x}, \bar{y}, \bar{z})$ be an optimal solution to the integer program IP with the original facility costs. And let $\bar{F}$, $\bar{C}$, $\bar{P}$ be the facility cost, connection cost, and rejecting penalties of solution $(\bar{x}, \bar{y}, \bar{z})$.

Notice that the facility cost only appears in the objective function in the linear program $LP - PRIMAL$. Thus, a feasible solution $(\bar{x}, \bar{y}, \bar{z})$ to $LP - PRIMAL$ with the original facility costs is also a feasible solution to $LP - PRIMAL$ with the scaled facility costs, and vice versa. Let $\bar{F}'$, $\bar{C}'$, $\bar{P}'$ be the facility cost, connection cost, and rejecting penalties of solution $(\bar{x}, \bar{y}, \bar{z})$ in the scaled instance. It is easy to see that

$$\bar{F}' = \lambda\bar{F}, \bar{C}' = \bar{C}, \bar{P}' = \bar{P}. \tag{12}$$

Let $S = (x', y', z')$ be the solution returned by the primal-dual method applied to the scaled instance. Let F', C', P' be the facility cost, connection cost, and rejecting penalties of solution in the scaled instance. Let F, C, P be the facility cost,connection cost, and rejecting penalties of solution (x', y', z') in the original instance. It is easy to see that

$$F' = \lambda F, P' = P, C' = C. \tag{13}$$

Lemma 2. $3\lambda F + P + C \leq 3(\lambda \bar{F} + \bar{P} + \bar{C})$.

Proof. Let $\alpha'_j, j \in \mathcal{C}$ be the resulted dual variables after applying the primal-dual method to the scaled instance. From Lemma 1, we know that $3F' + C' + 3P' \leq \sum_{j\in\mathcal{C}} \alpha'_j$. From the weak duality theorem, we know that the primal objective value of a feasible solution to the linear program $LP - PRIMAL$ is no less than any dual objective value. Thus, we have $\sum_{j\in\mathcal{C}} \alpha'_j \leq \bar{F}' + \bar{P}' + \bar{C}'$. Combining the above two inequalities, we have $3F' + C' + 3P' \leq 3(\bar{F}' + \bar{P}' + \bar{C}')$. Therefore we have $3F' + P' + C' \leq 3(\bar{F}' + \bar{P}' + \bar{C}')$. The lemma follows if we plug equations (12) and (13) into the above inequality.

2.4 Further Improvement by Local Search Technique

In this subsection, we will show how to further improve the quality of solution S by iteratively adding facilities to the existing initial solution. From Lemmas 1 and 2 we know that, to further improve the performance ratio, we need to reduce the connection cost (or equivalently increase the coefficient of C in the inequality $3F + 3P + C \leq 3\sum_{j\in\mathcal{C}} \alpha_j$). To reach this goal, our main idea is to trade connection cost with facility cost by iteratively opening more facilities. Clearly, adding additional facilities into the solution of the scaled instance returned by the primal-dual method increases the facility cost, and meanwhile decreases the connection cost and rejecting penalties. Thus when the increase on the facility cost by adding some additional facilities is less than the decrease on the connection cost and rejecting penalties, the quality of solution is improved and consequently the performance ratio is reduced. Hence, to make this approach work, we only need to answer the following questions.

1. What is the condition under which there will always exist some facility whose addition to the solution improves the quality?
2. If multiple facilities exist for improving the quality of solution, what would be the best order for selecting those facilities so as to minimize the total cost?
3. What is the final performance ratio after adding all possible facilities?

Below we discuss our idea for each of the three questions. We start with question 1. Let (x', y', z') be the solution generated by the primal-dual method on the scaled instance. Then we have the following lemma.

Lemma 3. *If* $(C + P) \leq (\bar{F} + \bar{P} + \bar{C})$, *then* $(F + P + C) \leq (2 - \frac{1}{3\lambda})\bar{F} + (1 + \frac{2}{3\lambda})(\bar{P} + \bar{C})$.

Proof. We rewrite the total cost $(F + P + C)$ as $\frac{3\lambda F+P+C}{3\lambda} + (1 - \frac{1}{3\lambda})(P + C)$. If $C + P \leq \bar{F} + \bar{P} + \bar{C}$, we have $(1 - \frac{1}{3\lambda})(P + C) \leq (1 - \frac{1}{3\lambda})(\bar{F} + \bar{P} + \bar{C})$. By Lemma 2, we have $\frac{3\lambda F+P+C}{3\lambda} \leq \frac{3(\lambda\bar{F}+\bar{P}+\bar{C})}{3\lambda}$. Adding the above two inequalities together, we have

$$(F + P + C) \leq (2 - \frac{1}{3\lambda})\bar{F} + (1 + \frac{2}{3\lambda})(\bar{P} + \bar{C}).$$

It is easy to see that the performance ratio could be better than 2 for some $\lambda > 1/3$ if the inequality $(C + P) \leq (\bar{F} + \bar{C} + \bar{P})$ holds. In fact, the performance ratio could even be 5/3 if set $\lambda = 1$. The best choice for λ depends also on other cases.

Lemma 4. *If there is a feasible solution S to the uncapacited facility location problem with penalties with facility cost F_S, rejecting penalties P_S, and connection cost C_S such that $(P_S + C_S) > (\bar{F} + \bar{P} + \bar{C})$, then there exists a facility whose addition to S improves the solution.*

Proof. First for each client j, we add into both S and the optimal solution OPT a dummy facility, $i_{penalty}$, with facility cost 0 and distance p_j/d_j. Let $\sigma(j)$ and $\bar{\sigma}(j)$ be j's closest facility in S and OPT respectively. Note that $\sigma(j)$ and $\bar{\sigma}(j)$ might be $i_{penalty}$.

Let $gain(i)$ be the change of the total cost caused by adding facility i into the solution S. Below we first determine the value of $gain(i)$. Clearly adding facility i into the solution increases the facility cost by f_i. For each client j connected to a facility $\sigma(j)$ in solution S, if $c_{ij} \leq c_{\sigma(j)j}$, we re-connect j to the newly opened facility i and decrease the connection cost by $d_j(c_{\sigma(j)j} - c_{ij})$; Otherwise the connection of j remains the same and has no change on the connection cost. Since this is the best way to reconnect clients after the addition of facility i, we have $gain(i) = \sum_{j \in \mathcal{C}: c_{ij} \leq c_{\sigma(j)j}} d_j(c_{\sigma(j)j} - c_{ij}) - f_i$.

Next we show the existence of facility i with $gain(i) > 0$ when $(P_S + C_S) > (\bar{F} + \bar{P} + \bar{C})$. We prove this by comparing the above approach we used for re-connecting clients to an added facility i with an imaginary approach which relates the current solution S to an optimal solution OPT. In the imaginary approach, for each added facility i to S, if $i \in OPT$, it re-connects a client j to i if and only if $\bar{\sigma}(j) = i$, regardless of the sign of $c_{\sigma(j)j} - c_{ij}$. Thus, $gain'(i) = \sum_{j \in \mathcal{C}: \bar{\sigma}(j) = i} d_j(c_{\sigma(j)j} - c_{ij}) - f_i$ for each $i \in OPT$.

We claim that $gain(i) \geq gain'(i)$ for each $i \in OPT$. This is because if $\bar{\sigma}(j) = i$ and $c_{\sigma(j)j} - c_{ij} < 0$, it contributes a negative term to $gain'(i)$ and a zero to $gain(i)$. If $\bar{\sigma}(j) = i$ and $c_{\sigma(j)j} - c_{ij} \geq 0$, it contributes the same term to both $gain'(i)$ and $gain(i)$. Also $gain(i)$ may contain some positive terms from those clients whose $\bar{\sigma}(j)$ is not equal to i. Thus, $gain(i)$ contains more positive terms and less negative terms than $gain'(i)$. Therefore, $gain(i) \geq gain'(i)$ for each $i \in OPT$.

Summing over all facilities in OPT, we have

$$\begin{aligned}
\sum_{i \in OPT} gain(i) &\geq \sum_{i \in OPT} gain'(i) \\
&\geq \sum_{i \in OPT} \sum_{j \in \mathcal{C}: \bar{\sigma}(j) = i} d_j(c_{\sigma(j)j} - c_{ij}) - f_i \\
&= \sum_{j \in \mathcal{C}} d_j c_{\sigma(j)j} - \sum_{j \in \mathcal{C}} d_j c_{\bar{\sigma}(j)j} - \sum_{i \in OPT} f_i \\
&= C_S + P_S - \bar{C} - \bar{P} - \bar{F}.
\end{aligned}$$

Thus, we obtain the following estimation for $gain(i)$

$$\sum_{i \in OPT} gain(i) \geq C_S + P_S - \bar{C} - \bar{P} - \bar{F}. \tag{14}$$

From the assumption of the lemma, we know that $C_S + P_S - \bar{C} - \bar{P} - \bar{F} > 0$. Hence, we have that $\sum_{i \in OPT} gain(i) > 0$ which implies that there is at least one facility i with $gain(i) > 0$.

The above lemma tells us that we can always find a facility whose addition to S improve the quality of solution unless $P_S + C_S \leq \bar{P} + \bar{C} + \bar{F}$. This suggests us to iteratively add unopened facility to the solution S so as to improve the performance ratio.

Next we discuss our idea for Question 2. That is, when multiple facilities exist for improving the quality of solution, what would be the order for adding them? Since we also want to bound the facility cost when we are adding facilities, a natural way is to consider the following simple greedy approach. The greedy approach considers the ratio of $gain(i)$ over f_i for each unopened facilities and adds the one with the largest gain ratio $(gain(i)/f_i)$ first. As we will show later that another advantage of using the greedy approach is that it enables us to derive a good estimation for the cost of the final solution by using inequality (14). The above local search heuristic can be summarized by the following pseudocode.

```
GREEDY-ADDING(S)
1  r ← 0
2  for each i ∈ F
3      do if gain(i)/f_i > r
4            then r ← gain(i)/f_i
5                 candidate ← i
6  if r > 0
7     then add candidate to S
8          GREEDY-ADDING(S)
9     else return
```

Next we discuss our idea for Question 3. The following lemma bounds the total cost of the final solution after executing the above heuristic.

Lemma 5. *Starting from a solution (x', y', z') with $C + P > \bar{C} + \bar{P} + \bar{F}$, the above greedy algorithm returns a solution with cost no more than $F + \bar{F} + \bar{P} + \bar{C} + \bar{F} \ln(1 + (C + P - \bar{F} - \bar{P} - \bar{C})/\bar{F})$.*

Proof. Let C_k, P_k, F_k be the connection cost, rejecting penalties, and facility cost after finishing the kth iteration. Since we start with the solution (x', y', z'), we have $C_0 = C$, $P_0 = P$, and $F_0 = F$. Let m be the smallest integer such that $C_m + P_m \leq \bar{C} + \bar{P} + \bar{F}$.

By Lemma 4, we know that when the above algorithm stops, $C_k + P_k \leq \bar{C} + \bar{P} + \bar{F}$, since otherwise there will be at least one facility i with $gain(i)/f_i > 0$. By the definition of m, we know that to prove this lemma, it is sufficient to show $F_m + C_m + P_m \leq F + \bar{F} \ln(1 + (C + P - \bar{F} - \bar{P} - \bar{C})/\bar{F}) + \bar{F} + \bar{P} + \bar{C}$.

First we note that if $C + P > \bar{C} + \bar{P} + \bar{F}$, then $\bar{F} > 0$. This is because if $\bar{F} = 0$, then $C + P = \bar{C} + \bar{P} + \bar{F}$. In this case, all clients are rejected.

By the definition of $\bar{F}$ and inequality (14), we have $\sum_{i \in OPT} f_i = \bar{F}$ and $\sum_{i \in OPT} gain(i) \geq P + C - \bar{F} - \bar{P} - \bar{C}$. By the pigeon hole principle, we know

that there exists at least one facility $i \in OPT$ with $gain(i)/f_i > (P_{k-1}+C_{k-1}-\bar{F}-\bar{P}-\bar{C})/\bar{F}$ in the kth iteration. Since the greedy algorithm adds a facility with the largest gain ratio among all facilities of $\mathcal{F}$ in the kth iteration for $1 \le k \le m$, in the k-th iteration the added facility has a gain ratio no less than $(P_{k-1}+C_{k-1}-\bar{F}-\bar{P}-\bar{C})/\bar{F}$. That means $(C_{k-1}+P_{k-1}-C_k-P_k-F_k+F_{k-1})/(F_k-F_{k-1}) \ge (P_{k-1}+C_{k-1}-\bar{F}-\bar{P}-\bar{C})/\bar{F}$, for $1 \le k \le m$. Rearranging the above inequality, we have $F_k - F_{k-1} \le \frac{(C_{k-1}+P_{k-1}-C_k-P_k)}{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}}, 1 \le k \le m$. Summing over all $1 \le k \le m$, we obtain $F_m - F_0 \le \sum_{1\le k\le m} \frac{C_{k-1}+P_{k-1}-C_k-P_k}{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}}$. Thus, we have $F_m + C_m + P_m \le F_0 + \sum_{1\le k\le m} \frac{C_{k-1}+P_{k-1}-C_k-P_k}{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}} + C_m + P_m$.

If we treat the right hand side of the above inequality as a function of variables C_m and P_m and treat C_k, P_k, $\bar{C}$, $\bar{P}$, and $\bar{F}$ as constants for $1 \le k \le m-1$, the right hand side is a linear function of C_m and P_m. And the term involving $C_m + P_m$ is $(1 - \frac{1}{1+(C_{m-1}+P_{m-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}})(C_m + P_m)$. From the defination of m, we know that $C_{m-1} + P_{m-1} - \bar{C} - \bar{P} - \bar{F} > 0$. Thus the coefficient of $C_m + P_m$ is positive and the right hand side assume its maximum value when $C_m + P_m = \bar{F} + \bar{C} + \bar{P}$ for all variables with $C_m + P_m \le \bar{F} + \bar{C} + \bar{P}$. Since we only want to get an upper bound for $F_m + C_m + P_m$, we can just use the equality $C_m + P_m = \bar{F} + \bar{C} + \bar{P}$.

We rewrite $\frac{C_{k-1}+P_{k-1}-C_k-P_k}{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}}$ as $\bar{F}(1-\frac{1+(C_k+P_k-\bar{C}-\bar{P}-\bar{F})/\bar{F}}{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}})$. Using the inequality $1 - x \le -\ln x$ for $x > 0$, and replacing x by $\frac{1+(C_k+P_k-\bar{C}-\bar{P}-\bar{F})/\bar{F}}{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}}$, we have

$$\begin{aligned} 1 - \frac{1+(C_k+P_k-\bar{C}-\bar{P}-\bar{F})/\bar{F}}{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}} &\le -\ln\left(\frac{1+(C_k+P_k-\bar{C}-\bar{P}-\bar{F})/\bar{F}}{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}}\right) \\ &= \ln\left(\frac{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}}{1+(C_k+P_k-\bar{C}-\bar{P}-\bar{F})/\bar{F}}\right). \end{aligned}$$

Note that $\frac{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}}{1+(C_k+P_k-\bar{C}-\bar{P}-\bar{F})/\bar{F}}$ is always positive since $C_k + P_k > \bar{C} + \bar{F} + \bar{P}$ for $1 \le k \le m-1$ and we use equality $C_m + P_m = \bar{C} + \bar{P} + \bar{F}$. Combining the above inequalities, we have

$$\begin{aligned} F_m - F_0 &\le \bar{F} \textstyle\sum_{k=1}^{m} \ln\left(\frac{1+(C_{k-1}+P_{k-1}-\bar{C}-\bar{P}-\bar{F})/\bar{F}}{1+(C_k+P_k-\bar{C}-\bar{P}-\bar{F})/\bar{F}}\right) \\ &= \bar{F} \textstyle\sum_{k=1}^{m} \{\ln(1 + (C_{k-1} + P_{k-1} - \bar{C} - \bar{P} - \bar{F})/\bar{F}) \\ &\quad - \ln(1 + (C_k + P_k - \bar{C} - \bar{P} - \bar{F})/\bar{F})\} \\ &\le \bar{F} \ln(1 + (C_0 + P_0 - \bar{C} - \bar{P} - \bar{F})/\bar{F}) \end{aligned}$$

where the last inequality is obtained by using the equality $C_m + P_m = \bar{F} + \bar{C} + \bar{P}$. Hence, we have $F_m + P_m + C_m \le \bar{F} \ln(1 + (C_0 + P_0 - \bar{C} - \bar{P} - \bar{F})/\bar{F}) + F_0 + P_m + C_m \le \bar{F} \ln(1 + (C_0 + P_0 - \bar{C} - \bar{P} - \bar{F})/\bar{F}) + F_0 + \bar{C} + \bar{P} + \bar{F}$, and the lemma follows.

With the above lemma, we are ready to analyze the performance ratio of the whole algorithm. Below is the two main steps of our algorithm.

1. Apply the primal-dual method on facility location problem with penalties with the scaled facility cost. Let S be the returned solution.

2. Run the iterative greedy heuristic *Greedy-Adding* on S and return the obtained solution.

Theorem 1. *There is an $O(n^3)$-time 1.8526-approximation algorithm for the uncapacited facility location problem with penalties.*

Proof. Let F be the facility cost, C be the connecting cost, and P be the rejecting penalties of the solution S obtained in the first phase. Let F^f be the facility cost, C^f be the connecting cost, and P^f be the rejecting penalties of the final solution. Depending on the solution S, we have two cases to consider: (1) $C+P \leq \bar{F}+\bar{C}+\bar{P}$ (2) or $C+P > \bar{F}+\bar{C}+\bar{P}$.

In case (1), the total cost satisfies $F+P+C \leq (2-\frac{1}{3\lambda})\bar{F}+(1+\frac{2}{3\lambda})(\bar{P}+\bar{C})$ by the Lemma 3. The cost of the final solution after running *Greedy-Adding* on S can not be worse than than $(2-\frac{1}{3\lambda})\bar{F}+(1+\frac{2}{3\lambda})(\bar{P}+\bar{C})$ since the algorithm adds a facility with a positive gain in each iteration.

For case (2), Lemma 5 ensures that the total cost of the final solution produced by the greedy heuristic satisfies $F^f+P^f+C^f \leq F+\bar{F}+\bar{C}+\bar{P}+\bar{F}\ln(1+(C+P-\bar{F}-\bar{P}-\bar{C})/\bar{F})$. By Lemma 2, we know that $C+P \leq 3\lambda\bar{F}-3\lambda F+3\bar{C}+3\bar{P}$. Combining the two inequalities, we get that $F^f+P^f+C^f \leq F+\bar{F}+\bar{C}+\bar{P}+\bar{F}\ln((3\lambda\bar{F}-3\lambda F+2\bar{C}+2\bar{P})/\bar{F})$. For any given instance, $\bar{F}$, $\bar{C}$, and $\bar{P}$ are fixed and can be treated as constant (even though we do not know their values). The right hand side of the above inequality can be viewed as a function of F which achieves its maximum $(1+\ln(3\lambda))\bar{F}+(1+\frac{2}{3\lambda})(\bar{C}+\bar{P})$ at $F=\frac{2}{3\lambda}(\bar{C}+\bar{P})$.

Therefore, the cost of the final solution is bounded by

$$\max\{(2-\frac{1}{3\lambda})\bar{F}+(1+\frac{2}{3\lambda})(\bar{P}+\bar{C}),(1+\ln(3\lambda))\bar{F}+(1+\frac{2}{3\lambda})(\bar{C}+\bar{P})\}$$

for both cases. Since $1+\ln(3\lambda) > 2-\frac{1}{3\lambda}$ for $\lambda > 1/3$, we have

$$\begin{aligned} F^f+C^f+P^f &\leq (1+\ln(3\lambda))\bar{F}+(1+\tfrac{2}{3\lambda})(\bar{C}+\bar{P}) \\ &\leq \max\{1+\ln(3\lambda),1+\tfrac{2}{3\lambda}\}(\bar{F}+\bar{C}+\bar{P}). \end{aligned}$$

Taking λ as the root of equation $\ln(3\lambda)=2/3\lambda$, which is $\lambda \simeq 0.7192$, the final cost is bounded by $1.8526(\bar{F}+\bar{P}+\bar{C})$.

As for the running time, the primal dual method in the first phase takes $O(n^2 \log n)$ time [3], where $n=|\mathcal{F}|+|\mathcal{C}|$. The *Greedy-Adding* heuristic in the second phase takes at most n iterations, and each iteration takes no more than $O(n^2)$ time since each $gain(i), i \in \mathcal{F}$ can be computed in $O(n)$ time. The total running time is thus bounded by $O(n^3)$.

References

1. V. Arya, N. Garg, A. Meyerson, K. Munagala, and V. Pandit, "Local Search Heuristics for k-median and Facility Location Problems," *Proc. 33rd ACM Symposium on Theory of Computing*, pages 21-29, 2001.

2. M. Charikar and S. Guha, "Improved combinatorial algorithms for facility location and k-median problems," *Proc. 39th IEEE Symposium on Foundations of Computer Science*, pages 378-388, 1999.
3. M. Charikar, S. Khuller, D. Mount, and G. Narasimhan, "Algorithms for Facility Location Problems with Outliers," *Proc. Symposium on Discrete Algorithms*, pages 642-651, 2001.
4. F. Chudak and D. Shmoys, "Improved approximation algorithms for a capacitated facility location problem," *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 875-876, 1999.
5. F. Chudak and D. P. Williamson, "Improved Approximation Algorithms for Capacitated Facility Location Problems," *Lecture Notes in Computer Science*, Vol. 1610, pages 99-113, 1999.
6. S. Guha and S. Khuller, "Greedy strikes back: Improved facility location algorithms," *J. Algorithms*, Vol. 31, No. 1, pages 228-248, 1999.
7. S. Guha and A. Meyerson and K. Munagala, "A constant factor approximation algorithm for the fault-tolerant facility location problem," *J. Algorithms* , Vol. 48, No. 2, pages 429-440, 2003.
8. K. Jain, and V. Vazirzni, "Approximation Algorithms for Metric Facility Location and k-Median Problems Using the Primal-Dual Schema and Lagrangian Relaxation," *J. ACM*, Vol. 48, No. 2, pages 274-296, 2001.
9. K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani, "Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP," *J. ACM*, Vol. 50, No. 6, pages 795-824, 2003.
10. M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Analysis of a local search heuristic for facility location problems," *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1-10, 1998.
11. J. -H. Lin and J. S. Vitter, "ϵ-approximation with minimum packing constraint violation," *Proceddings of the 24th Annual ACM Symposium on Theory of Computing*, pages 771-782, 1992.
12. D.B. Shmoys, "Approximation algorithms for facility location problem," *Approximation Algorithms for Combinatorial Optimization, Lecture Notes in Computer Science 1913, (K. Jansen and S. Khuller, eds.)*, Springer, Berlin, pages 27-33, 2000.
13. D. B. Shmoys, É. Tardos, and K. Aardal, "Approximation algorithms for facility location problems," *Proc. 29th ACM Symposium on Theory of Computing*, pages 265-274, 1997.
14. G. Xu and J. Xu, "An LP Rounding Algorithm for Approximating Uncapacitated Facility Location Problem with Penalties," *Inform. Proc. Lett.*, Vol. 94, No. 3, pages 119-123, 2005.
15. J. Zhang, "Approximating the two-level facility location problem via a quasi-greedy approach," *Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms* , pages 808 - 817, 2004.

The Reverse Greedy Algorithm for the Metric *K*-Median Problem

Marek Chrobak[1,⋆], Claire Kenyon[2], and Neal E. Young[1]

[1] Department of Computer Science, University of California, Riverside, CA 92521
[2] Computer Science Department, Brown University, Providence, RI 02912

Abstract. The Reverse Greedy algorithm (RGreedy) for the k-median problem works as follows. It starts by placing facilities on all nodes. At each step, it removes a facility to minimize the resulting total distance from the customers to the remaining facilities. It stops when k facilities remain. We prove that, if the distance function is metric, then the approximation ratio of RGreedy is between $\Omega(\log n/\log\log n)$ and $O(\log n)$.

Keywords: Analysis of algorithms, approximation algorithms, online algorithms.

1 Introduction

An instance of the *metric k-median problem* consists of a metric space $\mathcal{X} = (X, c)$, where X is a set of points and c is a *distance* function (also called the *cost*) that specifies the distance $c_{xy} \geq 0$ between any pair of nodes $x, y \in X$. The distance function is reflexive, symmetric, and satisfies the triangle inequality. Given a set of points $F \subseteq X$, the cost of F is defined by $cost(F) = \sum_{x \in X} c_{xF}$, where $c_{xF} = \min_{f \in F} c_{xf}$ for $x \in X$. Our objective is to find a k-element set $F \subseteq X$ that minimizes $cost(F)$.

Intuitively, we think of F as a set of facilities and of c_{xF} as the cost of serving a customer at x using the facilities in F. Then $cost(F)$ is the overall service cost associated with F. The k-element set that achieves the minimum value of $cost(F)$ is called the *k-median* of $\mathcal{X}$.

The k-median problem is a classical facility location problem and has a vast literature. Here, we review only the work most directly related to this paper. The problem is well known to be NP-hard, and extensive research has been done on approximation algorithms for the metric version. Arya *et al.* [1] show that the optimal solution can be approximated in polynomial time within ratio $3 + \epsilon$, for any $\epsilon > 0$, and this is the smallest approximation ratio known. Earlier, several approximation algorithms with constant, but somewhat larger approximation ratios appeared in the works by Charikar *et al.* [3], Charikar and Guha [2], and Jain and Vazirani [8]. Jain *et al.* [7] show a lower bound of $1 + 2/e$ on the approximation ratio for this problem (assuming P≠NP).

In the online version of the k-median problem, studied by Mettu and Plaxton [10], the algorithm is not given k in advance. Instead, requests for additional

⋆ Research supported by NSF Grant CCR-0208856.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 654–660, 2005.

facilities arrive over time. When a request arrives, a new facility must be added to the existing set. In other words, the algorithm computes an increasing sequence of facility sets $F_1 \subset F_2 \subset \cdots \subset F_n$, where $|F_k| = k$ for all k. The algorithm presented by Mettu and Plaxton [10] guarantees that $cost(F_k)$ approximates the optimal k-median cost within a constant factor (independent of k.) They also show that in this online setting no algorithm can achieve approximation ratio better than $2 - 2/(n-1)$.

The naive approach to the median problem is to use the greedy algorithm: Start with $F_0 = \emptyset$, and at each step $k = 1, \ldots, n$, let $F_k = F_{k-1} \cup \{f_k\}$, where $f_k \in X - F_{k-1}$ is chosen so that $cost(F_k)$ is minimized. Clearly, this is an online algorithm. It is not difficult to show, however, that its approximation ratio is $\Omega(n)$.

Reverse Greedy. Amos Fiat [5] proposed the following alternative idea. Instead of starting with the empty set and adding facilities, start with all nodes being facilities and remove them one by one in a greedy fashion. More formally, Algorithm RGreedy works as follows: Initially, let $R_n = X$. At step $k = n, n-1, \ldots, 2$, let $R_{k-1} = R_k - \{r_k\}$, where $r_k \in R_k$ is chosen so that $cost(R_{k-1})$ is minimized. For the purpose of online computation, the sequence of facilities can be precomputed and then produced in order $(r_1, r_2, \ldots, r_n)$.

Fiat [5] asked whether RGreedy is an $O(1)$-approximation algorithm for the metric k-median problem. In this note we present a nearly tight analysis of RGreedy by showing that its approximation ratio is between $\Omega(\log n / \log\log n)$ and $O(\log n)$. Thus, although its ratio is not constant, RGreedy performs much better than the forward greedy algorithm.

2 The Upper Bound

Fix k and let M be the optimal k-median of $\mathcal{X}$. Consider a step j of RGreedy (when we remove r_j), for $j > k$. Denote by Q the set of facilities in R_j that serve M. More specifically, Q is the smallest subset of R_j such that $c_{\mu Q} = c_{\mu R_j}$ for all $\mu \in M$. We estimate first the incremental cost in step j:

$$cost(R_{j-1}) - cost(R_j) \le \min_{r \in R_j - Q} cost(R_j - \{r\}) - cost(R_j) \tag{1}$$

$$\le \frac{1}{|R_j - Q|} \sum_{r \in R_j - Q} [cost(R_j - \{r\}) - cost(R_j)] \tag{2}$$

$$\le \frac{1}{j-k} \sum_{r \in R_j - Q} [cost(R_j - \{r\}) - cost(R_j)] \tag{3}$$

$$\le \frac{1}{j-k} [cost(Q) - cost(R_j)] \tag{4}$$

$$\le \frac{2}{j-k} cost(M). \tag{5}$$

The first inequality follows from the definition of R_{j-1}, in the second one we estimate the minimum by the average, and the third one follows from $|Q| \le k$. We now justify the two remaining inequalities.

Inequality (4) is related to the the super-modularity property of the cost function. We need to prove that

$$\sum_{r \in R-Q} [cost(R - \{r\}) - cost(R)] \leq cost(Q) - cost(R),$$

where $R = R_j$. To this end, we examine the contribution of each $x \in X$ to both sides. The contribution of x to the right-hand side is exactly $c_{xQ} - c_{xR}$. On the left-hand side, the contribution of x is positive only if $c_{xQ} > c_{xR}$ and, if this is so, then x contributes only to one term, namely the one for the $r \in R - Q$ that serves x in R (that is, $c_{xr} = c_{xR}$). Further, this contribution cannot be greater than $c_{xQ} - c_{xR}$ because $Q \subseteq R - \{r\}$. (Note that we do not use here any special properties of Q and R. This inequality holds for any $Q \subset R \subseteq X$.)

We now prove (5). Denote again $R = R_j$. For any $x \in X$, choose $r \in R$ and $\mu \in M$ that serve x in R and M, respectively. In other words, $c_{xR} = c_{xr}$ and $c_{xM} = c_{x\mu}$. We have $c_{\mu r} \geq c_{\mu Q}$, by the definition of Q. Thus $c_{xQ} \leq c_{x\mu} + c_{\mu Q} \leq c_{x\mu} + c_{\mu r} \leq 2c_{x\mu} + c_{xr} = 2c_{x\mu} + c_{xR}$. Now (5) follows by summation over all $x \in X$.

We have thus proved that $cost(R_{j-1}) - cost(R_j) \leq \frac{2}{j-k} cost(M)$. Summing up over $j = n, n-1, \ldots, k+1$, we obtain our upper bound.

Theorem 1. *The approximation ratio of Algorithm* RGreedy *in metric spaces is at most* $2H_{n-k} = O(\log n)$.

3 The Lower Bound

In this section we construct an n-point metric space $\mathcal{X}$ where, for $k = 1$, the ratio between the cost of the RGreedy's facility set and the optimal cost is $\Omega(\log n / \log\log n)$. (For general k, a lower bound of $\Omega(\log(n/k)/\log\log(n/k))$ follows easily, by simply taking k copies of $\mathcal{X}$.)

To simplify presentation, we allow distances between different points in $\mathcal{X}$ to be 0. These distances can be changed to some appropriately small $\epsilon > 0$ without affecting the asymptotic ratio. Similarly, whenever convenient, we will break the ties in RGreedy in our favor.

Let $\hat{T}$ be a graph that consists of a tree T with root ρ and a node μ connected to all leaves of T. T itself consists of h levels numbered $1, 2, \ldots, h$, with the leaves at level 1 and the root ρ at level h. Each node at level $j > 1$ has $(j+1)^3$ children in level $j - 1$.

To construct $\mathcal{X}$, for each node x of T at level j we create a cluster of $w_j = j!^3$ points (including x itself) at distance 0 from each other. Node μ is a 1-point cluster. All other distances are defined by shortest-path lengths in $\hat{T}$.

First, we show that, for $k = 1$, RGreedy will end up with the facility at ρ. Indeed, RGreedy will first remove all but one facility from each cluster. Without loss of generality, let those remaining facilities be located at the nodes of $\hat{T}$, and from now on we will think of w_j as the weight of each node in layer j. At the next step, we break ties so that RGreedy will remove the facility from μ.

We claim that in any subsequent step t, if j is the first layer that has a facility, then RGreedy has a facility on each node of T in layers $j+1,\ldots,h$. To prove it, we show that this invariant is preserved in one step. If a node x in layer j has a facility then, by the invariant, this facility serves all the nodes in the subtree T_x of T rooted at x, plus possibly μ (if x has the last facility in layer j.) What facility will be removed by RGreedy at this step? The cost of removing any facility from layers $j+1,\ldots,h$ is at least w_{j+1}. If we remove the facility from x, all the nodes served by x can switch to the parent of x, so the increase in cost is bounded by the total weight of T_x (possibly plus one, if x serves μ.) T_x has $(j+1)!^3/(i+1)!^3$ nodes in each layer $i \le j$. So the total weight of T_x is

$$\begin{aligned} w(T_x) &= \sum_{i=1}^{j} w_i \cdot (j+1)!^3/(i+1)!^3 \\ &= (j+1)!^3 \sum_{i=1}^{j} (i+1)^{-3} \\ &< (j+1)!^3 \\ &= w_{j+1}, \end{aligned}$$

where the inequality above follows from $\sum_{i=1}^{j}(i+1)^{-3} \le \sum_{i=2}^{\infty} i^{-2} < 1$. Thus removing x increases the cost by at most $w(T_x)+1 \le w_{j+1}$, so RGreedy will remove x or some other node from layer j in this step, as claimed. Therefore, overall, after $n-1$ steps, RGreedy will be left with the facility at ρ.

By the previous paragraph, the cardinality (total weight) of $\mathcal{X}$ is $n = w(T)+1 \le (h+1)!^3$, so $h = \Omega(\log n/\log\log n)$. The optimal cost and the cost of RGreedy are, respectively,

$$\begin{aligned} \mathit{cost}(\mu) &= \sum_{i=1}^{h} i \cdot w_i \cdot (h+1)!^3/(i+1)!^3 \\ &= (h+1)!^3 \sum_{i=1}^{h} i(i+1)^{-3} \\ &\le (h+1)!^3 \sum_{i=2}^{\infty} i^{-2} \\ &< (h+1)!^3, \quad \text{and} \\ \mathit{cost}(\rho) &= \sum_{i=1}^{h} (h-i) \cdot w_i \cdot (h+1)!^3/(i+1)!^3 \\ &= (h+1)!^3 \sum_{i=1}^{h} (h-i)(i+1)^{-3} \\ &\ge (h-1)(h+1)!^3/8, \end{aligned}$$

where in the last step we estimate the sum by the first term. Thus the ratio is $\mathit{cost}(\rho)/\mathit{cost}(\mu) \ge (h-1)/8 = \Omega(\log n/\log\log n)$.

In the argument above we considered only the case $k = 1$. More generally, one might characterize the performance ratio of the algorithm as a function of both n and k. Any lower bound for $k = 1$ implies a lower bound for larger k by simply taking k (widely separated) copies of the metric space. Therefore we obtain:

Theorem 2. *The approximation ratio of Algorithm* RGREEDY *in metric spaces is not better than* $\Omega(\log(n/k)/\log\log(n/k))$.

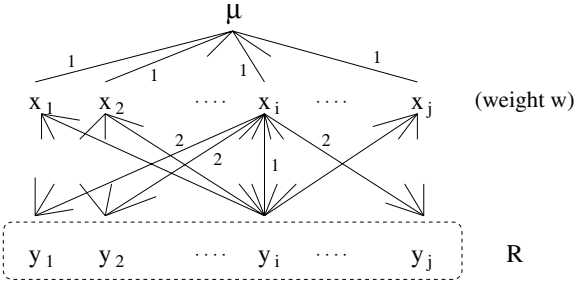
4 Technical Observations

We have shown an $O(\log n)$ upper bound and an $\Omega(\log n/\log\log n)$ lower bound on the approximation ratio of RGREEDY for k-medians in metric spaces. Next we make some observations about what it might take to improve our bounds.

Comments on the Upper Bound. We focus on the case $k = 1$. In the upper bound proof in Section 2 we show that the incremental cost of RGREEDY when removing r_j from R_j to obtain R_{j-1} is at most $2cost(\mu)/(j-1)$. The proof of that fact (inequalities (1) through (5)) uses very little information about the structure of R_j. In fact, that proof shows that for *any* set R of size j,

$$\min_r cost(R - \{r\}) - cost(R) \le \frac{2cost(\mu)}{j-1}. \tag{6}$$

Next we describe a set R of size j in a metric space for which this latter bound is tight. The metric space is defined by the following weighted graph:



The space has points $\mu, x_1, \ldots, x_j, y_1, \ldots, y_j$, where the points x_i have weights w, for some large integer w. (In other words, each x_i represents a cluster of w points at distance 0 from each other.) All other points have weight 1. Point μ is connected to each x_i by an edge of length 1. Each x_i is connected to y_i by an edge of length 1, and to each y_l, for $l \neq i$, by an edge of length 2.

For $k = 1$, the optimal cost is $cost(\mu) = j(w+2)$. Now consider $R = \{y_1, \ldots, y_j\}$. Removing any $y_i \in R$ increases the cost by $w \approx cost(\mu)/j$. Thus, for this example, inequality (6) is tight, up to a constant factor of about 2.

Of course, RGreedy would not produce the particular set R assumed above for R_j. Instead, RGreedy would leave a facility in each cluster x_i. Also, this example only shows a *single iteration* where the incremental cost matches the upper bound. (It is possible to modify this example to show that this bound can be tight in any constant number of steps, but the constant factor by which the bound is off increases with the number of steps.) Nonetheless, the example demonstrates that to improve the upper bound it is necessary to consider some information about the structure of R_j (due to the previous steps of RGreedy). Perhaps one could use an amortized analysis, bound the incremental cost of *groups* of steps, or use an indirect charging scheme. It may also be possible to use a better bound on $cost(M)$ (e.g. by linear programming duality.)

Comments on the Lower Bound. We can show that the lower-bound constructions similar to that in Section 3 are unlikely to give any improvement. The formal statement of this claim is rather technical, and is included here only for the benefit of readers who might be interested in continuing this work.

Fix a metric space X with n points. Let μ be the 1-median of X, and assume (by scaling) that its cost is $cost(\mu) = n/2$. Let B be the unit ball around μ, that is the set of points at distance at most 1 from μ. Note that $|B| \geq n/2$.

For $i \geq 1$, define Q_i to be the points $x \in X$ such that $i - 1 \leq c_{x\mu} \leq i$, and there is a time when x is used by RGreedy as a facility for some point in B. (In particular, $Q_1 = B$.) Let h be the maximum index for which $Q_h \neq \emptyset$. Define t_i to be the last time at which RGreedy had a facility in $Q_1 \cup Q_2 \cup \cdots \cup Q_i$, and let m_i be the number of points served by Q_i at time t_{i-6}. (The value of 6 is not critical; any $c \geq 6$ will work, with some minor modifications.)

Lemma 1. *Suppose that* $\sum_{i=10}^{h} im_i = O(n)$. *Then, for* $k = 1$, RGreedy*'s approximation ratio is* $O(\log n / \log\log n)$.

Proof sketch: We will show that $h = O(\log n / \log\log n)$. Since the facility computed by RGreedy at step n is at distance at most h from μ, this will imply the lemma.

Consider any $i \leq h - 9$. One can show that in steps $t_i, t_i + 1, \ldots, t_{i+3}$, RGreedy's cost to serve B increases by at least $n/2$, while all facilities that serve B at steps $t_i + 1, \ldots, t_{i+3}$ are in $Q_{i+1} \cup \ldots \cup Q_{i+5}$. Thus, at some step t' with $t_i \leq t' \leq t_{i+3}$, RGreedy's incremental cost is at least $(n/2)/(1 + |Q_{i+1} \cup Q_{i+2} \cup \cdots \cup Q_{i+5}|)$.

Suppose $Q_{i+9} \neq \emptyset$. Before step t', deleting *all* facilities in Q_{i+9} would have increased the cost by $O(im_{i+9})$, so for some facility in Q_{i+9} its deletion would have increased the cost by $O(im_{i+9}/|Q_{i+9}|)$. Since RGreedy did not delete this facility at time t', we have (by this and the previous paragraph)

$$(n/2)/(1 + |Q_{i+1} \cup Q_{i+2} \cup \cdots \cup Q_{i+5}|) = O(im_{i+9}/|Q_{i+9}|).$$

Rewriting and summing the above over i (including now those i for which Q_{i+9} is empty),

$$\sum_{i=1}^{h-9} \frac{|Q_{i+9}|}{1+|Q_{i+1}\cup Q_{i+2}\cup\cdots\cup Q_{i+5}|} = O\Big(\frac{1}{n}\sum_{i=1}^{h-9} i m_{i+9}\Big)$$
$$= O\Big(\frac{1}{n}\sum_{i=10}^{h} i m_i\Big) \;=\; O(1).$$

From this and $\sum_{i=1}^{h}|Q_i| \le n$ one can show that $h = O(\log n/\log\log n)$. □

Note that lemma applies to the space used in Section 3. In this case, each set Q_i consists of the nodes in T at level i, and $m_i = (h+1)!^3/(i+1)^3$ is the total weight of level i so, indeed, $\sum_{i=1}^{h} i m_i = O(h!^3) = O(n)$. When RGreedy is applied in this space, at each time t_i, the facilities that serve the nodes at distance at most i from μ are concentrated in two layers. The lemma indicates that in order to improve the bound, one needs to design an example where these facilities are always distributed more or less uniformly across the remaining facilities.

Acknowledgments

We would like to thank Amos Fiat, Christoph Dürr, Jason Hartline, Anna Karlin, and John Noga for useful discussions.

References

1. V. Arya, N. Garg, R. Khandekar, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. In *Proc. 33rd ACM Symposium on Theory of Computing*, pages 21–29, 2001.
2. M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proc. 40th IEEE Symposium on Foundations of Computer Science*, pages 378–388, 1999.
3. M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proc. 31st ACM Symposium on Theory of Computing*, pages 1–10, 1999.
4. U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of ACM*, 45(4):634–652, 1998.
5. A. Fiat. Private communication.
6. D. S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22:148–162, 1982.
7. K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proc. 34th ACM Symposium on Theory of Computing*, pages 731–740, 2002.
8. K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of ACM*, 48:274–296, 2001.
9. J.-H. Lin and J. S. Vitter. e-approximations with minimum packing constraint violation (extended abstract). In *Proc. 24th ACM Symposium on Theory of Computing*, pages 771–782, 1992.
10. R. Mettu and C. Plaxton. The online median problem. *SIAM Journal on Computing*, 32:816–832, 2003.
11. N. E. Young. K-medians, facility location, and the Chernoff-Wald bound. In *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 86–95, 2000.

On Approximate Balanced Bi-clustering

Guoxuan Ma, Jiming Peng*, and Yu Wei

Department of Computing and Software McMaster University
Hamilton, Ontario L8S 4K1, Canada
{mag3,pengj,weiy3}@mcmaster.ca

Abstract. We consider the balanced bi-clustering problem for a given data set, where the number of entities in each cluster is bounded, and its special case where the number of entities in each cluster is fixed. Several algorithms to attack these problems are proposed. In particular, a novel and efficient heuristic, in which we first reformulate the constrained bi-clustering problem into a quadratic programming(QP) problem and then try to solve it by optimization technique, is proposed. We prove that our algorithm can provide a 2-approximate solution to the original problem. Promising numerical results are reported.

1 Introduction

In general, clustering refers to the problem of partitioning a given set $\mathcal{S} = \{s_1, \cdots, s_n\}$ into several groups based on some similarity measurement. Clustering has many important applications in various areas such as image segmentation, object recognition, information retrieval and market analysis [10]. Bi-clustering involves with partitioning the data set into two clusters. It is the basis of the so-called hierarchical divisive clustering. Bi-clustering has been well-studied by expertise in the communities of computational geometry and graph theory [2, 5, 9, 11].

In many applications, finding clusters that satisfy user-specified constraints is highly desirable. This leads to the so-called constrained clustering first introduced in [3]. An important case of constraint-based clustering is the so-called balanced clustering, where the size of each cluster is bounded. A special case of balanced clustering is when the cardinality of every cluster is fixed. Both cases have applications in market analysis and the design of wireless sensor networks.

Among various clustering methods, the classical K-Means is probably the most popular and widely used. K-Means usually aims at minimizing the Sum-of-Squared (MSSC) distance from each entity to its assigned cluster center

$$\min \sum_{k=1}^{K} \sum_{s_i \in C_k} (s_i - c_k)^2 \tag{1}$$

* This work was supported by the NSERC discovery grant # 249635-02 of Canada, a PREA award of Ontario and the MITACS project "New Interior Point Methods and Software for Convex Conic-Linear Optimization and Their Application to Solve VLSI Circuit Layout Problems".

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 661–670, 2005.

where $c_k = \sum_{i \in C_k} s_i/|C_k|$ is the centroid of cluster C_k and $|C_k|$ is the cardinality of C_k. Although widely adopted in partitioning clustering, K-Means also has some drawbacks. For example, in most cases K-means can not find the global minimum of the measurement in the model, it is very sensitive to initialization methods and prone to be trapped at a local optimum. In case of balanced clustering, the classical K-Means is not applicable.

In [4], Bradley *et al* proposed to use linear optimization technique to solve the subproblem in their model for constrained K-Means. In the present paper, we use the same theoretical framework as that in [4]. Nevertheless, by restricting us to the balanced bi-clustering, we propose a simple heuristics to find the optimal solution of the subproblem and show that our simple heuristics enjoys a lower complexity than the approach in [4].

Secondly, we consider the issue of how to find an approximate solution to the balanced bi-clustering. Our algorithm is based on a similar idea as in [5]. Further, for the special case where the sizes of cluster are fixed, we propose a new method (Q-Means), which could not only give a 2-approximation bound theoretically, but also enjoy a lower iteration bound than our algorithm for general balanced bi-clustering. Our preliminary experiments show that our new method outperforms the popular K-Means for the test problems in term of both the running time and the quality of the solution.

The paper is organized as follows. In section 2, we first introduce the so-called balanced bi-clustering problem and propose several algorithms, including a heuristic and an approximation algorithm with an upper bound. In Section 3, we focus on fixed size bi-clustering problem. We also construct an algorithm via reformulating the problem to a Quadratic Programming(QP) problem. Preliminary computational results are reported in Section 4.

2 Algorithms for Balanced Bi-clustering

2.1 Improved Constrained K-Means Algorithm for Bi-clustering

Mathematically, balanced bi-clustering can be defined as the following constrained optimization problem:

$$\min_{x_{ij}} \sum_{j=1}^{2} \sum_{i=1}^{n} x_{ij} \left\| s_i - \frac{\sum_{l=1}^{n} x_{lj} s_l}{\sum_{l=1}^{n} x_{lj}} \right\|^2 \tag{2}$$

$$S.T. \quad \sum_{j=1}^{2} x_{ij} = 1 \quad (i = 1, \cdots, n) \tag{3}$$

$$\sum_{i=1}^{n} x_{ij} \geq \tau_j \quad (j = 1, 2) \tag{4}$$

$$x_{ij} \geq 0 \quad (i = 1, \cdots, n; j = 1, 2) \tag{5}$$

The constrained K-Means algorithm proposed by Bradley et'al [4] can be described as follows.

Constrained K-Means Algorithm

S.0 **Initialization:** $t = 0$, starting cluster centers c_1^t and c_2^t at iteration t;

S.1 Compute c_1^{t+1} and c_2^{t+1} at iteration $t + 1$ in following 2 steps:

1.1 **Cluster Assignment:** Given (c_1^t, c_2^t), solve the following subproblem for x_{ij}^t

$$\begin{aligned} &\min_{x_{ij}} \sum_{j=1}^{2} \sum_{i=1}^{n} x_{ij} \left\| s_i - c_j^t \right\|^2 \\ &S.T. \qquad \text{constraints (3)-(5)} \end{aligned} \tag{6}$$

1.2 **Cluster Update:** Update C_1^t and C_2^t as follows:

$$c_j^{t+1} = \begin{cases} \frac{\sum_{l=1}^{n} x_{lj} s_l}{\sum_{l=1}^{n} x_{lj}} & \text{If } \sum_{i=1}^{n} x_{i,j}^t > 0, \\ c_j^t & \text{Otherwise.} \end{cases} \tag{7}$$

S.2 Stop when $c_j^{t+1} = c_j^t$, $j = 1, 2$, otherwise increase t by 1 and go to step 1.

It has been showed in [4] that the constrained K-Means will eventually terminate in a finite number of iterations at a cluster assignment that is locally optimal such that $x_{ij} \in \{0, 1\}$. The time complexity of this algorithm depends on the algorithm that is chosen for the sub-problem (6). In [4], fast network simplex algorithms are suggested for solving the subproblem, which enjoys an $O(n \log^2 n)$ complexity according to [1].

In the sequel, we propose a simple heuristic for the balanced bi-clustering problem. Given c_1^t and c_2^t, a partition of the date set, C_1^t and C_2^t, is derived. If C_1^t and C_2^t satisfy the constraints in (6), then we are done; Otherwise, we can use the following rounding procedure to extract a solution that satisfies the constraints in (6). Without loss of gentility, suppose we need to move some entities from C_1 to C_2:

Rounding Procedure 1

(1) For every $s_i \in C_1$
Evaluate the function $f(s_i) = \|s_i - c_2\| - \|s_i - c_1\|$ for s_i;

(2) Sort all the entities based on $f(s_i)$, move $(|C_1| - \tau_1)$ entities which have the least value of $f(s_i)$ to C_2 to satisfy the constraints.

This procedure is essentially a greedy algorithm with a $O(n \log n)$ complexity. We now claim that the rounding procedure 1 provides us a partitioning that satisfies the constraints in (6), which is an optimal solution to (6). To see this, suppose to the contrary that the solution provided by the rounding procedure is not optimal. Then, we can reduce the objective function in (6) by moving some entities from C_2 to C_1, or swapping two entities in C_1 and C_2. This is definitely impossible as either moving one entity from C_2 to C_1 or swapping two entities in C_1 and C_2 will lead to an increase of the objective function.

Therefore, by using the rounding procedure 1 to solve the subproblem (6), we can improve the constrained K-Means for balanced bi-clustering slightly. The complexity of this algorithm is $O(Tn \log n)$, where T is the times that we run the algorithm.

In what follows, we introduce another rounding procedure. Note that if we move an entity s_i from C_1 to C_2, the centroids of these two clusters after the movement will be changed to [13]

$$c_1 \leftarrow \frac{|C_1|c_1 - s_i}{|C_1| - 1}; \quad c_2 \leftarrow \frac{|C_2|c_2 + s_i}{|C_2| + 1}. \tag{8}$$

Correspondingly, the objective function value becomes

$$v(s_i) = \frac{|C_2|}{|C_2| + 1} \|c_2 - s_i\|^2 - \frac{|C_1|}{|C_1| - 1} \|c_1 - s_i\|^2, \quad s_i \in C_1. \tag{9}$$

Using the above relations, we obtain the following rounding procedure.

Rounding Procedure 2
Repeat:
For every $s_i \in C_1$, calculate $v(s_i)$ by (9);
Move the entity with minimal $v(s_i)$ from C_1 to C_2, and update the cluster centroids by (8);
Until the constraints in (2) are satisfied.

The complexity of this procedure is $O(n^2)$. The rounding procedure 2 can also be applied after the constrained K-Means with rounding procedure 1 stops. In such a situation, the rounding procedure 2 serves as a local search procedure to further reduce the objective function. This two-phase heuristic is similar to HK-Means [7] for unconstrained clustering.

2.2 An Approximation Algorithm

In the sequel, we consider an approximation algorithm for the balanced case. Our algorithm follows a similar idea as in [5] for problem (1). Let us consider the following partitioning problem:

$$\begin{aligned} \min_{C_1, C_2, c_1 \in C_1, c_2 \in C_2} & \sum_{s_i \in C_1} \|s_i - c_1\|^2 + \sum_{s_i \in C_2} \|s_i - c_2\|^2 \\ S.T. \ & |C_1| \geq \tau_1, \quad |C_2| \geq \tau_2 \quad (\tau_1 + \tau_2 \leq n). \end{aligned} \tag{10}$$

In other words, we impose the requirement that the centroids of the two clusters must be chosen from entities in the corresponding clusters. We have

Lemma 1. *The constrained bi-clustering problem (10) for n entities in any fixed dimension d can be solved in $O(n^3 \log n)$ time.*

Proof. We can enumerate all pair of entities in the data set as the (c_1, c_2) in (10). For every fixed pair of (c_1, c_2), we can use the step 1 in Algorithm 3 to find a solution for problem (10). Since every run of such a procedure takes $O(n \log n)$ time and in total we have $n(n-1)/2$ pairs, the constrained problem (10) can be solved in $O(n^3 \log n)$ time.

Lemma 2. *Given a data set $\mathcal{S}$ with $s_i \in \Re^d$, centered at $c = \frac{1}{n}\sum_{s_i \in \mathcal{S}} s_i$. If a vector $s \in \Re^d$ satisfies*

$$\|s - c\| \leq \|s_i - c\|, \quad \forall i = 1, \cdots, n,$$

then we have

$$\sum_{s_i \in \mathcal{S}} \|s_i - s\|^2 \leq 2 \sum_{s_i \in \mathcal{S}} \|s_i - c\|^2.$$

Proof. It is straightforward to see that

$$\begin{aligned}
\sum_{s_i \in \mathcal{S}} \|s_i - s\|^2 &= \sum_{s_i \in \mathcal{S}} \|s_i - c + c - s\|^2 \\
&= \sum_{s_i \in \mathcal{S}} \|s_i - c\|^2 + |\mathcal{S}|\|c - s\|^2 + \sum_{s_i \in \mathcal{S}} (s_i - c)^T (c - s) \\
&= \sum_{s_i \in \mathcal{S}} \|s_i - c\|^2 + |\mathcal{S}|\|c - s\|^2 \leq 2 \sum_{s_i \in \mathcal{S}} \|s_i - c\|^2,
\end{aligned}$$

where the last inequality follows from the assumption in the lemma.

A direct consequence of the above lemma is

Lemma 3. *The sum-of-squared error at the optimal solution of the constrained problem (10) is at most twice as large as the sum-of-squared error at the optimal solution of problem (2).*

Proof. Let $\{(c_1, C_1), (c_2, C_2)\}$ be an optimum solution of (2). For such a fixed partition (C_1, C_2), we consider the following minimization problems

$$\min_{s \in C_i} \|s - c_i\| \quad i = 1, 2. \tag{11}$$

Denote the solutions of the above two problems by c_1' and c_2' respectively. From Lemma 2, we obtain

$$\sum_{s \in C_1} \|c_1' - s\|^2 + \sum_{s \in C_2} \|c_2' - s\|^2 \leq 2 \sum_{s \in C_1} \|c_1 - s\|^2 + 2 \sum_{s \in C_2} \|c_2 - s\|^2 \tag{12}$$

Recall the fact that the partition (C_1, C_2) satisfy the constraints in problem (2) and thus (c_1', C_1, c_2', C_2) is also a feasible solution of problem (10). This implies that the sum-of-squared error of the optimal solution to problem (10) is less than or equal to the sum-of-squared error based on the centers obtained by solving problem (11), which further concludes the lemma.

Based on the above analysis, we propose the following algorithm.

Algorithm 1: Approximation Algorithm for Balance Bi-clustering

(1) For every pair of (c_1, c_2) where $c_1, c_2 \in \mathcal{S}$
Use (c_1, c_2) as the starting centers and apply the rounding procedure 1 to solve problem (2).

(2) Output the bipartition with smallest sum-of-squares error as a solution.

Combining Lemma 1 and Lemma 3, we have

Theorem 4. *For a given set of n entities, Algorithm 1 can provides a 2-approximate solution to problem (2), and the time complexity of the algorithm is* $O(n^3 \log n)$.

3 Approximation Method to Fixed Size Bi-clustering

In this section, we consider the fixed size bi-clustering problem defined by

$$\begin{aligned} &\min_{C_1,C_2} \textstyle\sum_{s_i\in C_1} \|s_i - c_1\|^2 + \sum_{s_i\in C_2} \|s_i - c_2\|^2 \\ &S.T. \qquad \tfrac{|C_1|}{|C_2|} = R \quad (R \geq 1) \end{aligned} \tag{13}$$

Here c_1 and c_2 are the geometry center of C_1 and C_2, respectively. For a given data set $\mathcal{S}$, let us denote its centroid by $\bar{c}$, i.e., $\bar{c} = \frac{1}{n}\sum_{i=1}^n s_i$. Suppose that we partition $\mathcal{S}$ into two clusters $(c_1, C_1), (c_2, C_2)$. W. l. o. g., we assume $|C_1| \geq |C_2| > 0$. It is easy to see that $(1-t)c_1 + tc_2 = \bar{c}$ for some $t \in (0, \frac{1}{2}]$, or equivalently

$$c_2 = c_1 + Q(\bar{c} - c_1), \quad Q = \frac{1}{t}. \tag{14}$$

It follows:

$$\|s_i - c_2\|^2 = \|s_i - c_1\|^2 + Q^2\|\bar{c} - c_1\|^2 - 2Q(s_i - c_1)^T(\bar{c} - c_1) \tag{15}$$

For every entity s_i, let us define an indicator function

$$\phi(c_1, s_i) = Q^2\|\bar{c} - c_1\|^2 - 2Q(s_i - c_1)^T(\bar{c} - c_1)$$

Consequently, we have $\|s_i - c_2\|^2 = \|s_i - c_1\|^2 + \phi(c_1, s_i)$. Using this notation, we can rewrite the MSSC model as the following bilevel optimization problem

$$\min_{c_1,Q} \sum_{i=1}^n (\|s_i - c_1\|^2 + \min\{0, \phi(c_1, s_i)\}).$$

For a given cluster center c_1 and fixed Q, we can determine the cluster that each entity belongs to by its indicator function $\phi(c_1, s_i)$. If $\phi(c_1, s_i) \geq 0$, then $s_i \in C_1$; otherwise, $s_i \in C_2$. Let us define the active index set $I = \{i : \phi(c_1, s_i) < 0\}$. Then the MSSC model can be transformed into a quadratic programming(QP) problem:

$$\min_{c_1} \sum_{i=1}^n \|s_i - c_1\|^2 + \sum_{i\in I} \phi(c_1, s_i). \tag{16}$$

From our above discussion, it becomes clear that solving problem (16) equals to assigning entities to two clusters whose centroids are c_1 and $c_1 + Q(\bar{c} - c_1)$, respectively. On the other hand, given a partition $(c_1, C_1), (c_2, C_2)$, one has

$$\bar{c} = \frac{|C_1|}{n}c_1 + \frac{|C_2|}{n}c_2$$

For problem (13), since $\frac{|C_1|}{|C_2|} = R$, we have $Q = R+1$. This implies if we know the cardinalities of the clusters at a solution, then we can decide what should be the value of the parameter Q used in problem (16). However, the parameter Q in the model (QP) can not determine precisely the sizes of two clusters. Nevertheless, we still expect that Q can help us to control the relative cardinality of the two clusters. This inspires us to propose the so-called Q-Means method described as follows:

Algorithm 2: Q-Means Algorithm

S.0 Given size ratio requirement R ($R \geq 1$);
S.1 **Initialization:** Choose an entity in the space as the starting center of cluster C_1, say c_1. Set $Q = R + 1, i = 1, c_1^i = c_1$.
S.2 **Iteration:**
Identify the active set $I = \{i : \phi(c_1^i, s_i) \leq 0\}$;
Solve problem (16) for c_1^{i+1} based on the current active index set I. Goto Step 2 if $c_1^i \neq c_1^{i+1}$;
S.3 **Rounding:** Use the rounding procedure 1 or 2 to find a solution that satisfies the size ratio constraint.
S.4 **Alternation:** Set $Q = 1 - 1/(R + 1)$, repeat step 2 and step 3.
S.5 **Output:** Output the better one from the two solutions obtained from these two procedures as the final solution.

In principle, Q-Means is a heuristic similar to K-Means. We iteratively reduce the objective function until convergence and then use a rounding procedure to find a solution that satisfies the constraint. Similar to K-means, Q-means is also very sensitive to the choice of initial starting entity and also very easy to be trapped in some local optimum after a few iterations. The complexity of Q-Means is $O(Tn \log n)$ if the rounding procedure 1 is applied, where T is the maximal number of iterations in running the algorithm. Alternatively, if we use rounding procedure 2 in Algorithm 2, then it will lead to a complexity of $O(Tn^2)$.

On the other hand, to get a good approximation solution, we employ the idea described in section 2.2. However, in the special case of (13), we just need to try every entity in the set as the starting center used in the Q-Means algorithm and then compute another center via the relation (14). This reduces the complexity of the whole procedure from $O(n^2)$ to $O(n)$ of enumerating all possible starting entities for the bi-clustering problems.

Algorithm 3: Revised Q-Means

S.1 Try every entity in the data set as the starting center in Algorithm 2;
S.2 Output the best bipartition (C_1^*, C_2^*) as a solution.

The complexity of this algorithm is $O(n^2 \log n)$ if rounding procedure 1 is used in every run of Q-Means, and $O(n^3)$ if rounding procedure 2 is used. In what follows we derive an upper bound for the solution produced by the revised Q-Means algorithm.

Theorem 5. *The sum-of-squared error of the solution derived by the revised Q-Means is at most twice as large as the sum-of-squared error of the optimum solution of problem (13).*

Proof. Let $\{(c_1, C_1), (c_2, C_2)\}$ be the optimal solution of the problem (13). Let us denote the solutions of the problems (11) by c_1' and c_2' respectively. Define

$$c_1^* := \frac{n}{|C_2|}\bar{c} - Rc_1' = c_2 + R(c_1 - c_1'),$$
$$c_2^* := \frac{n}{|C_1|}\bar{c} - \frac{1}{R}c_2' = c_1 + \frac{1}{R}(c_2 - c_2').$$

It follows

$$\|c_1^* - c_2\| = R\|c_1 - c_1'\|, \quad \|c_2^* - c_1\| = \frac{1}{R}\|c_2 - c_2'\|.$$

From the above two relations, we must have either

$$\|c_1^* - c_2\| \le \|c_2' - c_2\|, \tag{17}$$

or

$$\|c_2^* - c_1\| \le \|c_1' - c_1\|. \tag{18}$$

W. l. o. g., we assume inequality (17) holds. By Lemma 2, we have

$$SSE := \sum_{s_i \in C_1} \|s_i - c_1'\|^2 + \sum_{s_i \in C_2} \|s_i - c_1^*\|^2 \le 2 \sum_{s_i \in C_1} \|s_i - c_1\|^2 + 2 \sum_{s_i \in C_2} \|s_i - c_2\|^2.$$

Recall the fact that the pair (c_1', c_1^*) had been used in the revised K-Means as the starting centers for two clusters. Therefore, the sum-of-square error provided by the final output from the algorithm must be less than or equal to SSE. Similarly, we can derive the same conclusion when inequality (18) holds. This finishes the proof of the theorem.

4 Computational Results

In this section, we present some preliminary computational results to illustrate the performance of the revised Q-Means, compared with the improved constrained K-Means. The performance of two rounding procedures is compared as well. The data sets that we used in the experiments include a synthetic data set and some well-known benchmarks in machine learning literature.

For fairness, we use the same starting strategy for the two algorithms, i.e., for both K-Means and Q-Means, we enumerate all the entities as one of the centroid and calculate the other one according to the size ratio constraint as we described in the procedure of Q-Means. Also, the same rounding procedure is applied in both K-Means and Q-Means. The running time and solution quality of two algorithms are compared.

We mention that all the tests in the current work are performed on a personal computer with a Pentium 4 1700 MHz Processor, with a 256M memory, the CPU time is in seconds.

We first use a random generator to produce various two-dimensional synthetic data sets approximately in the mixture Gaussian distribution [8], which are recognized as one of best test beds for MSSC clustering. Four data sets containing 4000 entities in [0,0] × [1,1] plane, are generated for our experiments, S01 with $v = 0.05, r = 0.0$, S02 with $v = 0.10, r = 0.2$, S03 with $v = 0.10, r = 0.4$, S04 with $v = 0.15, r = 0.4$. (v means variance and r stands for noise level.)

The following result is obtained by using rounding procedure 1 in both algorithms, the size ratio is fixed as 1:3 for these four data sets.

data set	v	r	K-Means (CPU time)	Q-Means (CPU time)
S01	0.05	0.0	9.121028 (14.432218)	9.121028 (9.513855)
S02	0.10	0.2	165.422969 (61.143902)	165.422969 (33.010462)
S03	0.10	0.4	280.028264 (80.566430)	280.028264 (38.851260)
S04	0.15	0.4	281.542429 (109.471963)	281.044463 (42.279831)

The numerical result based on rounding procedure 2 is reported below.

data set	v	r	K-Means(CPU time)	Q-Means(CPU time)
S01	0.05	0.0	9.121028 (13.834134)	9.121028 (8.171109)
S02	0.10	0.2	165.354702 (117.635497)	165.354702 (93.065645)
S03	0.10	0.4	278.773878 (287.669128)	278.769649 (242.697759)
S04	0.15	0.4	279.253263 (340.284469)	279.207675 (272.663927)

We have also tested the Q-Means algorithm on the the following test problems in literature. The first problem is relatively small, therefore, the rounding procedure 2 is used. For the second data set, which is relatively large, we apply the rounding procedure 1 instead.

– **Soybean Data Set (large)** from the UCI Machine Learning Repository[1], see also [12]. This data set has 307 instances and each instance has 35 normalized attributes.

The Soybean Data (large)

size ratio	K-Means (CPU time)	Q-Means (CPU time)
1:1	4558.433495 (4.225763)	4558.433495 (1.594678)
1:2	4517.974430 (3.419375)	4517.974430 (1.409188)
1:3	4478.362896 (0.897566)	4478.362896 (0.326273)
1:4	4543.901639 (0.331163)	4543.901639 (0.085307)

– **Spam E-mail Database** created by M. Hopkins *et al.* It has 4601 samples, 57 attributes (we remove the last boolean attribute which indicates whether the e-mail was consider spam or not). We obtain the following result for this data set:

Spam E-mail Database

size ratio	K-Means (CPU time)	Q-Means (CPU time)
1:1	1609292966.72 (2060.0481)	1586877787.41 (301.9538)
1:2	1405187739.99 (1807.5085)	1404622481.26 (334.4043)
1:3	1276474668.00 (1800.1751)	1276437654.01 (334.0490)
1:4	1276437654.01 (1565.5531)	1187542869.25 (311.2299)

[1] http://www.ics.uci.edu/~mlearn/MLRepository.html

The above table illustrates that if only rounding procedure 1 is used and the data set is large, then the difference between K-Means and Q-Means might be substantial.

It should be pointed out that for the first experiment, since rounding procedure 2 is applied, most CPU time is spent on the rounding process, therefore, the difference between K-Means and Q-Means is not that significant compared with the case when rounding procedure 1 is applied.

From the above experiments we can conclude that, at least for these data sets, Q-Means has outperformed regular K-Means in terms of both running time and solution quality. This is not surprising, as we pointed out earlier, besides reducing the complexity of enumerating all the entities, Q-Means uses the constraint to guide the search process rather than to satisfy the constraint passively as in the rounding K-Means.

References

1. Aggarwal. C, Kaplan, H., Orlin J. and Tarjan. R, A Faster Primal Network Simplex Algorithm. Operations Research Center, Massachusetts Institute of Technology. OR 315-96. (1996)
2. Asano, T. Effective Use of Geometry properties for clustering. JCDCG'98, LNCS 1763, (2000) 30-46
3. Batagelj, V. and Ferligoj, A. Constrained Clustering Problems. Proc. of IFCS'98. (1998)
4. Bradley, P., Bennet, K. and Demiriz, A., Constrained K-Means Clustering. MSR-TR-2000-65, Microsoft Research. (2000)
5. Hasegawa, S., Imai, H., Inaba, M. Efficient Algorithms for Variance-Based k-clustering. First Pacific Conf. on Comp. Graph. and Appl. (1993)
6. Hansen, P. and Jaumard, B. Clustering analysis and mathematical programming. Math. Prog. B., 79(1997) 191-215
7. Hansen, P. and Mladenovic, N. J-means: a new local search heuristic for minimum sum-of-squares clustering. Pattern Recog., 34(2001) 405-413
8. He, J., Lan, M., Tan, C., Sung, S. and Low, H. Initialization of clusters refinement algorithms: a review and comparative study. *International Joint Conf. on Neural Networks*, IJCNN, (2004) 25-29
9. Inaba, M., Katoh, N. and Imai, H. Applications of Weighted Voronoi Diagrams and Randomization to Variance-Based k-clustering. Proc. of 10th ACM Symp. on Comp. Geo., (1994) 332-339
10. Jain, A.K., Murty, N.M. and Flynn, P.J.(1999) Data Clustering: A Review. ACM Comput. Surveys, 31(1999) 264-323
11. Matoušek, J.(2000) On Approximate Geometric k-clustering. Dis. and Comput. Geo., 24(2000) 61-84
12. Michalski, R.S. and Chilausky, R.L. Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems*, 4(2), 1980 125-161
13. Späth, H. *Cluster analysis Algorithms for Data Reduction and Classification of Objects*, John Wiley and Sons, Ellis Horwood Ltd, (1980)
14. Tung, A., Ng, R., Lakshmanan, L. and Han, J.,(2001) Constraint-Based Clustering in Large Databases. Proc. of the 8th Inter. Conf. on Database Theory, (2001) 405-419

Toroidal Grids Are Anti-magic

Tao-Ming Wang

Department of Mathematics
Tung-Hai University, Taichung, Taiwan 40704, R.O.C.
wang@mail.thu.edu.tw

Abstract. An anti-magic labeling of a finite simple undirected graph with p vertices and q edges is a bijection from the set of edges to the integers $\{1, ..., q\}$ such that all p vertex sums are pairwise distinct, where the vertex sum on a vertex is the sum of labels of all edges incident to such vertex. A graph is called anti-magic if it has an anti-magic labeling. Hartsfield and Ringel [3] conjectured that all connected graphs except K_2 are anti-magic. Recently, N. Alon et al [1] showed that this conjecture is true for p-vertex graphs with minimum degree $\Omega(\log p)$. They also proved that complete partite graphs except K_2 and graphs with maximum degree at least $p-2$ are anti-magic. In this article, some new classes of anti-magic graphs are constructed through Cartesian products. Among others, the toroidal grids $C_m \times C_n$(the Cartesian product of two cycles), and the higher dimensional toroidal grids $C_{m_1} \times C_{m_2} \times ..... \times C_{m_t}$, are shown to be anti-magic. Moreover, the more general result is also proved to be true: $H \times C_n$ (hence $C_n \times H$) is anti-magic, where H is an anti-magic k-regular graph, where $k > 1$.

1 Introduction

All graphs in this paper are finite undirected simple graphs with no loops. In 1989, Hartsfield and Ringel[3] introduced the concept so called anti-magic graph. An anti-magic labeling of a graph $G = (V, E)$ with p vertices and q edges is a bijection $f: E \to \{1, ..., q\}$ such that the induced vertex sum $f^+: V \to \mathbf{Z}^+$ is injective, where $f^+(u) = \sum\{f(uv)|uv \in E\}$. A graph is called anti-magic if it has an anti-magic labeling. Hartsfield and Ringel showed that paths, cycles, complete graphs K_p $(p > 2)$ are anti-magic. They conjectured that all trees except K_2 are anti-magic. Further, every connected graph besides K_2 is anti-magic. These two conjectures are unsettled. Recently, Alon et al [1] validated that the last conjecture is true for all graphs with p (> 4) vertices and minimum degree $\Omega(\log p)$. They also proved that if G is a graph with p (> 4) vertices and $\Delta(G) > p-2$, then G is anti-magic, and that all complete partite graphs except K_2 are anti-magic.

In this paper we consider the anti-magicness of Cartesian products of graphs. Using these constructions we may construct new classes of anti-magic graphs. For more conjectures and open problems on anti-magic graphs and various graph labeling problems, please see the dynamic survey article of Gallian [2].

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 671–679, 2005.

2 Anti-magic Labelings of Toroidal Grids

The Cartesian product $G_1 \times G_2$ of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the graph with vertex set $V_1 \times V_2$, and $u = (u_1, u_2)$ is adjacent with $v = (v_1, v_2)$ whenever $u_1 = v_1$ and $u_2v_2 \in E_2$, or, $u_2 = v_2$ and $u_1v_1 \in E_1$. A toroidal grid graph is the Cartesian product of two cycles. Before we state the main result about the anti-magic labelings of toroidal grid graphs, let us first put two lemmas which will show the anti-magicness of paths and cycles:

Lemma 1. *All paths P_{m+1} are anti-magic for integers $m \geq 2$.*

Proof:

Suppose the vertex set is $\{v_1, ..., v_{m+1}\}$ and the edge set is particularly arranged to be $\{v_iv_{i+2} | i = 1, ..., m-1\} \cup \{v_mv_{m+1}\}$. And an anti-magic edge labeling can be assigned as $f(v_iv_{i+2}) = i$, for $1 \leq i \leq m-1$, and $f(v_mv_{m+1}) = m$. Note that the edge labeling induces the following ordering on vertices as

$$f^+(v_1) < f^+(v_2) < ..... < f^+(v_{m+1})$$

since the vertex sums are

$$f^+(v_i) = \begin{cases} i & \text{if } i = 1, 2; \\ 2i-2 & \text{if } i = 3, ..., m; \\ 2m-1 & \text{if } i = m+1. \end{cases}$$

Q.E.D.

Note that the anti-magic labeling of C_m can be easily deduced from that of P_{m+1} by "identifying" vertices v_1 and v_2 in above proof. Hence we have:

Lemma 2. *All cycles C_m are anti-magic for integers $m \geq 3$.*

Proof:

In the proof of last lemma by identifying v_1 and v_2, and moving the index of the rest vertices 1 forward in P_{m+1}, we have the vertex set $\{v_1, ..., v_m\}$ and the edge set $\{v_1v_2, v_1v_3\} \cup \{v_iv_{i+2} | i = 2, ..., m-2\} \cup \{v_{m-1}v_m\}$ for C_m. And an anti-magic edge labeling can be assigned as $f(v_1v_2) = 1$, $f(v_1v_3) = 2$, $f(v_iv_{i+2}) = i+1$, for $2 \leq i \leq m-2$, and $f(v_{m-1}v_m) = m$. Note that the edge labeling induces the following ordering on vertices as

$$f^+(v_1) < f^+(v_2) < ..... < f^+(v_m)$$

since the vertex sums are

$$f^+(v_i) = \begin{cases} 3 & \text{if } i = 1; \\ 2i & \text{if } i = 2, ..., m-1; \\ 2m-1 & \text{if } i = m. \end{cases}$$

Q.E.D.

Theorem 1. *The toroidal grid graphs $C_m \times C_n$ are anti-magic for integers $m, n \geq 3$.*

Proof:

Let $f: E(C_m \times C_n) \to \{1, 2, ..., 2mn\}$ be an edge labeling of $C_m \times C_n$, and the induced vertex sum at the vertex (u, v) is $f^+(u, v) = \sum f((u, v), (y, z))$, where the sum is running over all vertices (y, z) adjacent to (u, v) in $C_m \times C_n$. Note that in the product graph $C_m \times C_n$, at each vertex (u, v) the edges incident to such vertex can be partitioned into two parts, one part is contained in one copy of C_m component, and the other part is contained in another copy of C_n component. Hence then in $C_m \times C_n$, let s and t be the restriction of f^+ on C_m component and C_n component respectively, i.e., $s(u, v) = \sum f((u, v), (y, v))$, where the sum is running over all vertices y adjacent to u in C_m, and $t(u, v) = \sum f((u, v), (u, z))$, where the sum is running over all vertices z adjacent to v in C_n. Therefore $f^+(u, v) = s(u, v) + t(u, v)$.

Now we claim that we can label the edges in $C_m \times C_n$ so that

$$f^+(u_1, v_1) < f^+(u_2, v_1) < ..... < f^+(u_m, v_1) <$$

$$f^+(u_1, v_2) < f^+(u_2, v_2) < ..... < f^+(u_m, v_2) <$$

$$.....$$

$$f^+(u_1, v_n) < f^+(u_2, v_n) < ..... < f^+(u_m, v_n)$$

and hence we have an anti-magic edge labeling.

It is well known that all cycles are anti-magic (Lemma 2). On each C_m component and C_n component, we may fix anti-magic labelings $L_1: E(C_m) \to \{1, 2, ..., m\}$, and $L_2: E(C_n) \to \{1, 2, ..., n\}$, which induce the distinct vertex sums, hence strict orderings, on vertex sums of C_m and C_n respectively (see Figure 1). Without loss of generality, we may rename the vertices of C_m and C_n such that $s(u_1, v) < s(u_2, v) < ..... < s(u_m, v)$ for each $v \in V(C_n)$, and $t(u, v_1) < t(u, v_2) < ..... < t(u, v_n)$ for each $u \in V(C_m)$.

To show the above claim, for the first step we label the edges on the i-th C_m component of $C_m \times C_n$ with vertices $(u_1, v_i), (u_2, v_i), ..., (u_m, v_i)$, for $i = 1, 2, ..., n$. Using the fixed edge labeling for the i-th C_m component, where $i = 1, 2, ..., n$, and

$$1, 2, ..., m, \text{ (edge labels for 1st } C_m \text{ component)}$$

$$m + 1, m + 2, ..., 2m, \text{ (edge labels for 2nd } C_m \text{ component)}$$

$$.....$$

$$(n - 1)m + 1, (n - 1)m + 2, ..., nm \text{ (edge labels for n-th } C_m \text{ component)}$$

then we have

$$s(u_1, v_1) < s(u_2, v_1) < ..... < s(u_m, v_1) <$$

$$s(u_1, v_2) < s(u_2, v_2) < ..... < s(u_m, v_2) <$$

$$.....$$

$$s(u_1, v_n) < s(u_2, v_n) < ..... < s(u_m, v_n)$$

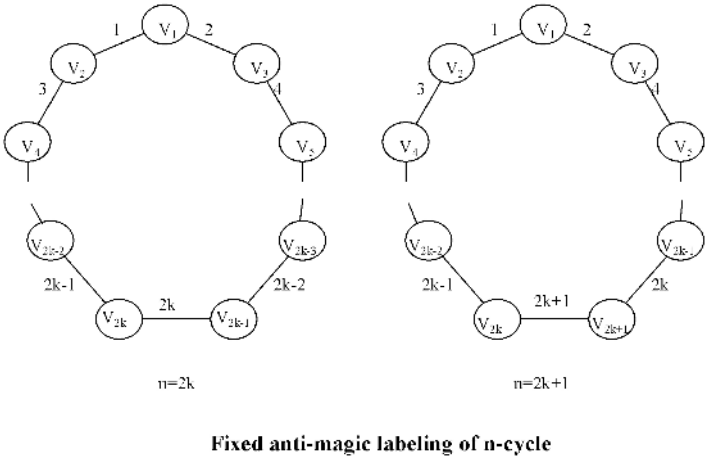


Fig. 1. Fixed anti-magic labels of C_m and C_n

Note first that it is obviously $s(u_1, v_i) < s(u_2, v_i) < ..... < s(u_m, v_i)$ for $i = 1, 2, ..., n$. On the other hand, $s(u_m, v_i) < s(u_1, v_{i+1})$ for $i = 1, 2, ..., n-1$, since among two groups of numbers $(i-1)m+1, (i-1)m+2, ..., im$ and $im+1, im+2, ..., (i+1)m$, adding any two in first group is always less than adding any two in the second group.

After labeling edges along C_m components as above, we continue the edge labeling along the second components, i.e., the C_n components. The following consecutive edge labeling is consisting of mixture of two orderings. The first is the fixed edge labeling order on C_n with either the usual order (U) which induces the strict ordering on vertices $t(v_1) < t(v_2) < ..... < t(v_n)$ (Figure 1), or the reversed order (R) on vertices $(u_1, v_i), (u_2, v_i), ..., (u_m, v_i)$, $i = 1, 2, ..., n$, and the second is the fixed vertex ordering $u_1, u_2, ..., u_m$ on C_m. The labeling goes like the following (see Figure 4 as an example):

$$nm+1, nm+2, ..., (n+1)m-1, (n+1)m, \text{ (in usual order)}$$

$$(n+2)m, (n+2)m-1, ..., (n+1)m+2, (n+1)m+1, \text{ (in reversed order)}$$

$$(n+3)m, (n+3)m-1, ..., (n+2)m+2, (n+2)m+1, \text{ (in reversed order)}$$

.....

etc.

and naturally we have the following three cases (see Figure 2) for finishing the labeling:

Case 1. n is even

The above edge labeling creates constant vertex sum along each C_n component of each vertex: $t(u_1, v_j) = t(u_2, v_j) = ... = t(u_m, v_j)$, for $1 \le j \le n$. Note that $t(u_m, v_i) < t(u_1, v_{i+1})$, for $i = 1, 2, ..., n-1$, since $t(u_1, v_{i+1}) = t(u_m, v_{i+1})$ and $t(u_m, v_i) < t(u_m, v_{i+1})$, for $i = 1, 2, ..., n-1$.

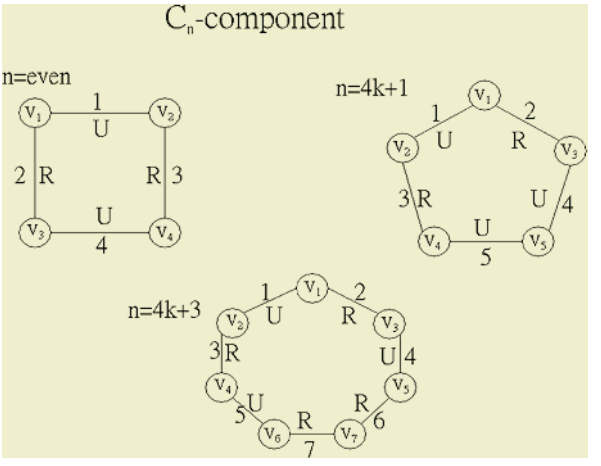


Fig. 2. Anti-magic labelings of cycles in usual(U) order, and in reversed(R) order

Case 2. $n \equiv 1 \pmod 4$

We have constant vertex sum along each C_n component of each vertex: $t(u_1, v_j) = t(u_2, v_j) = ... = t(u_m, v_j)$, for $1 \leq j \leq n-1$. Note that $t(u_m, v_i) < t(u_1, v_{i+1})$, for $i = 1, 2, ..., n-2$, since $t(u_1, v_{i+1}) = t(u_m, v_{i+1})$ and $t(u_m, v_i) < t(u_m, v_{i+1})$, for $i = 1, 2, ..., n-2$. As for $j = n$, both edges at each of the vertices $(u_1, v_n), (u_2, v_n), ..., (u_m, v_n)$ are assigned in the same usual order. Thus $t(u_1, v_n) < t(u_2, v_n) < ..... < t(u_m, v_n)$. At vertices (u_m, v_{n-1}) and (u_1, v_n), we see that $t(u_m, v_{n-1}) < t(u_1, v_n)$, since $t(u_1, v_{n-1}) = t(u_m, v_{n-1})$ and $t(u_m, v_{n-1}) < t(u_m, v_n)$. The reason why $t(u_m, v_{n-1}) < t(u_m, v_n)$ is that for vertices (u_m, v_{n-1}) and (u_m, v_n), they share one edge label, and the other edge label of (u_m, v_{n-1}) is less than that of (u_m, v_n).

Case 3. $n \equiv 3 \pmod 4$

Similar to the above case, except $j = n$, we have $t(u_1, v_j) = t(u_2, v_j) = ... = t(u_m, v_j)$, for $1 \leq j \leq n-1$. As for $j = n$, both edges at each of the vertices $(u_1, v_n), (u_2, v_n), ..., (u_m, v_n)$ are assigned the same reversed order, and we modify it to switch the usual order to reversed one, and switch the reversed order to the usual one (see Figure 3). Then it reduces to the situation similar to case 2.

Therefore in each of above three cases, we have

$$t(u_1, v_1) \leq t(u_2, v_1) \leq ..... \leq t(u_m, v_1) \leq$$

$$t(u_1, v_2) \leq t(u_2, v_2) \leq ..... \leq t(u_m, v_2) \leq$$

$$.....$$

$$t(u_1, v_n) \leq t(u_2, v_n) \leq ..... \leq t(u_m, v_n)$$

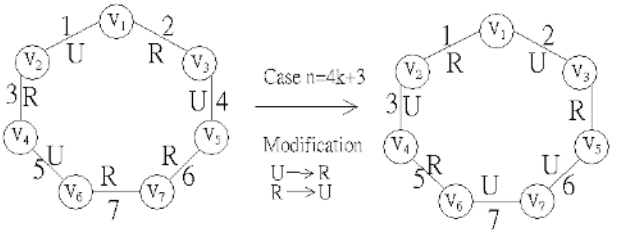


Fig. 3. Modification of the labeling of C_n in case $n \equiv 3 \pmod 4$

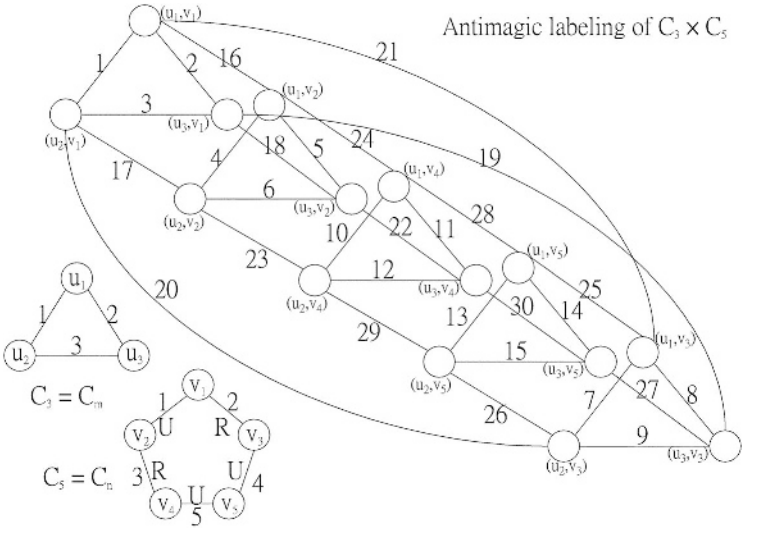


Fig. 4. An Example of Anti-Magic Labeling of $C_m \times C_n$

Thus along with the previous observations on vertex sums restricted on s, we have that

$$f^+(u_1, v_1) < f^+(u_2, v_1) < ..... < f^+(u_m, v_1) <$$
$$f^+(u_1, v_2) < f^+(u_2, v_2) < ..... < f^+(u_m, v_2) <$$
$$.....$$
$$f^+(u_1, v_n) < f^+(u_2, v_n) < ..... < f^+(u_m, v_n)$$

and the vertex sums are hence distinct, therefore the toroidal grids $C_m \times C_n$ are anti-magic for $m, n \geq 3$, as desired. **Q.E.D**

Please see an explicit example (figure 4) of an anti-magic labeling for $C_m \times C_n$.

Notice that cycles are special cases of regular graphs, therefore it is natural to consider the more general situation using regular graphs instead of cycles.

The above result can be generalized to the following in a quite straightforward fashion:

Theorem 2. *$H \times C_n$ (hence $C_n \times H$) is anti-magic, where H is an anti-magic k-regular graph ($k > 1$) on m vertices, and C_n is a cycle on n vertices for $n \geq 3$.*

Proof:

We will show $H \times C_n$, the Cartesian product of H and C_n, is anti-magic for integers $m, n \geq 3$. Let $f^+ : V(H \times C_n) \to N$ be the vertex sum induced from the edge labeling $f : E(H \times C_n) \to \{1, 2, ..., mn(1 + k/2)\}$ on $H \times C_n$, i.e., $f^+(u, v) = \sum f((u, v), (y, z))$, where the sum is running over all vertices (y, z) adjacent to (u, v) in $H \times C_n$. Note that in the product graph $H \times C_n$, at each vertex (u, v) the edges incident to such vertex can be partitioned into two parts, one part is contained in one copy of H component, and the other part is contained in one copy of C_n component. Hence then in $H \times C_n$, let s and t be the restriction of f^+ on H component and C_n component respectively, i.e., $s(u, v) = \sum f((u, v), (y, v))$, where the sum is running over all vertices y adjacent to u in H, and $t(u, v) = \sum f((u, v), (u, z))$, where the sum is running over all vertices z adjacent to v in C_n. Therefore $f^+(u, v) = s(u, v) + t(u, v)$.

Now we claim that we can label the edges in $H \times C_n$ so that

$$f^+(u_1, v_1) < f^+(u_2, v_1) < ..... < f^+(u_m, v_1) <$$

$$f^+(u_1, v_2) < f^+(u_2, v_2) < ..... < f^+(u_m, v_2) <$$

$$.....$$

$$f^+(u_1, v_n) < f^+(u_2, v_n) < ..... < f^+(u_m, v_n)$$

Since H and C_n are anti-magic, on each H component and C_n component, we may fix anti-magic labelings $1, 2, ..., \frac{mk}{2}$, and $1, 2, ..., n$, which induce the distinct vertex sums, hence strict orderings, on vertex sums of H and C_n respectively (see Figure 1). Without loss of generality, we may rename the vertices of H and C_n such that $s(u_1) < s(u_2) < ..... < s(u_m)$ and $t(v_1) < t(v_2) < ..... < t(v_n)$, where s and t, by abusing the language, are the vertex sums induced from the edge labelings of H and C_n respectively.

To show the above claim, for the first step we label the edges of each H component of $H \times C_n$ (with vertices $(u_1, v_i), (u_2, v_i), ..., (u_m, v_i)$ for the i-th H component, $i = 1, 2, ..., n$) using the fixed edge labeling as follows:

$$1, 2, ..., m,$$

$$m + 1, m + 2, ..., 2m,$$

$$.....$$

$$(n - 1)m + 1, (n - 1)m + 2, ..., nm$$

then we have

$$s(u_1, v_1) < s(u_2, v_1) < ..... < s(u_m, v_1) <$$
$$s(u_1, v_2) < s(u_2, v_2) < ..... < s(u_m, v_2) <$$
$$.....$$
$$s(u_1, v_n) < s(u_2, v_n) < ..... < s(u_m, v_n)$$

Note first that it is obviously $s(u_1, v_i) < s(u_2, v_i) < ..... < s(u_m, v_i)$ for $i = 1, 2, ..., n$. On the other hand, $s(u_m, v_i) < s(u_1, v_{i+1})$ for $i = 1, 2, ..., n-1$, since H is regular, and among two groups of numbers $(i-1)m+1, (i-1)m+2, ..., im$ and $im+1, im+2, ..., (i+1)m$, adding any k numbers in first group is always less than adding any k numbers in the second group.

After labeling edges along H components as above, we continue the edge labeling along the second components, i.e., the C_n component, exactly as in the proof of previous theorem.

Therefore we have

$$t(u_1, v_1) \leq t(u_2, v_1) \leq ..... \leq t(u_m, v_1) \leq$$
$$t(u_1, v_2) \leq t(u_2, v_2) \leq ..... \leq t(u_m, v_2) \leq$$
$$.....$$
$$t(u_1, v_n) \leq t(u_2, v_n) \leq ..... \leq t(u_m, v_n)$$

Thus along with the previous observations on vertex sums restricted on s, we have that

$$f^+(u_1, v_1) < f^+(u_2, v_1) < ..... < f^+(u_m, v_1) <$$
$$f^+(u_1, v_2) < f^+(u_2, v_2) < ..... < f^+(u_m, v_2) <$$
$$.....$$
$$f^+(u_1, v_n) < f^+(u_2, v_n) < ..... < f^+(u_m, v_n)$$

and the vertex sums are hence distinct, therefore $H \times C_n$ is anti-magic for $n \geq 3$ and H is an anti-magic k-regular graph $(k > 1)$ as desired. **Q.E.D**

Corollary 1. *The higher dimensional toroidal grid graph $C_{m_1} \times C_{m_2} \times ..... \times C_{m_t}$ is anti-magic, where the integers $m_1, m_2, ..., m_t \geq 3$, and $t \geq 2$.*

Proof:

Note that $C_{m_1} \times C_{m_2} \times ..... \times C_{m_{t-1}}$ is regular and anti-magic by induction and Theorem 2. And the Corollary follows. **Q.E.D**

3 Future Studies

To conclude the article, the following are two similar open problems related to the current article for further exploration, among others:

1. All prism graphs $C_m \times P_n$ are anti-magic.
2. All lattice grid graphs $P_m \times P_n$ are anti-magic.

References

1. N. Alon, G. Kaplan, A. Lev, Y. Roditty and R. Yuster, *Dense graphs are anti-magic,* Journal of Graph Theory, 47(2004), pp. 297-309.
2. J.A. Gallian, *A dynamic survey of graph labeling,* The Electronic J. of Combin., (2001), DS6, pp. 1-79.
3. N. Hartsfield and G. Ringel, *Pearls in Graph Theory,* Academic Press Inc., Boston, 1990 (Revised version, 1994), pp.108-109.

Optimally Balanced Forward Degree Sequence

Xiaomin Chen, Mario Szegedy, and Lei Wang

Department of Computer Science, Rutgers University

Abstract. Forward degree seqences, arising from orderings of the vertices in a graph, carry a lot of vital information about the graph. In this paper, we focus our work on two special classes of forward degree sequences, which we named balanced and strongly balanced. Our main result is to prove that any chordal graph has a strongly balanced forward degree sequence and any graph with all degrees at most 3 has a balanced forward degree sequence. Moreover, we show that the (strongly) balanced forward degree sequence can be computed in polynomial time in the above cases.

1 Introduction

A company has two open positions. On the waiting list there are n almost equally good candidates. Some pairs of the candidates can work together, some pairs can not. These pairs are known to the company. Most likely a person will accept the offer as soon as he gets it. It is also possible that he gets some better job and turns our offer down. We want to offer these candidates one at a time and get an immediate response. The goal is to maximize the likelihood of hiring two candidates that can work together. In what order the company should give the offers to the candidates?

Questions of this kind and its variants are the starting point of our research in this article. We refer to these problems as *offer rejection problem*. Our goal is to find the best strategy dealing with the possible rejections.

In search for answers to the offer rejection problem, we studied a subject that is interesting of its own right, namely, the *forward degree sequences of graphs*. A forward degree sequence arises from an ordering σ of the vertices of a graph. We eliminate the vertices according to this ordering, and the forward degree d_v^σ of a vertex v is its degree in the remaining graph when we eliminate it.

The idea of forward degree sequences is related to two classical topics in graph theory, namely, the *degree sequence* and *vertex elimination order*. The degree sequences of graphs are well characterized in [2], [3], and [4]. The vertex elimination order gives a nice characterization of chordal graphs. (See [7].) The forward degree sequences we define and study here arise from very different questions and are of different nature.

There are several nice connections between the offer rejection problem and the forward degree sequences. However, in this paper we focus ourselves to the pure graph-theoretical study of forward degree sequences. We mention one of

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 680–689, 2005.

the connections here. We associate to each forward degree sequence a polynomial $P_\sigma(z) = \sum_{v \in V} z^{d_v^\sigma}$. We find that the offer rejection problem with rejection probability q is equivalent to the problem of finding the σ which minimizes $P_\sigma(1/q)$ in the graph with edges represent pairs that can not work together. We will define an ordering σ to be more *balanced* than τ if $P_\sigma(1/q) \leq P_\tau(1/q)$ for all probability q. A related notion we define is *strongly balanced*. It is an interesting combinatorial question whether a graph has a most balanced (strongly balanced) forward degree sequence. We prove that this includes some nice classes of graphs. For example, chordal graphs and 3-regular graphs, where we also give polynomial time algorithm to find the most balanced sequences.

The forward degree sequences carry a lot of information of their graphs. One may easily express some usual graph parameters in terms of properties about the forward degree sequences. (See Section 2.) In Section 3 we define some new graph parameters based on the forward degree sequences. These parameters, besides their close relation to the offer rejection problem, are of pure graph-theoretic interest as well. One of the interesting problems that remains open is how to compute some of the parameters in polynomial time. Our main result, states that every 3-regular graph has a most balanced forward degree sequence, gives some new insight to the graph isomorphism problem ([5] and [6]).

1.1 Notations

All the graphs we consider in this article are undirected simple graphs. For any graph G, $E(G)$ is the set of edges in G, $V(G)$ is the set of vertices in G. The complement of G is denoted by $\overline{G}$. For any $S \subseteq V$, $G[S]$ is the subgraph of G induced by S. For any $v \in V$, the degree of v in G is denoted by d_v^G and the neighbors of v denoted by $N_G(v)$. The induced subgraph of G by deleting v is denoted by $G - v$. If $x, y \in V$, and xy is an edge in G, $G - xy$ is the graph G with edge xy deleted. If xy is not an edge in G, $G + xy$ is the graph gotten by adding the edge xy to G.

Let $S = (s_1, \cdots, s_n)$ be a sequence. We denote by $S(i)$ the i-th element of S. The concateneation of S and a new element, $(s_1, \cdots, s_n, v)$, is denoted by (S, v); and the concateneation of a new element and S, $(v, s_1, \cdots, s_n)$, is denoted by (v, S); $S[i \leftrightarrow j]$ is the sequence we get from S by exchanging the elements on the i-th position and the j-th position. We view a permutation on an n element set as a sequence of length n.

If S is a sequence of integers, we denote by $N_S(k)$ the number of occurrences of k in S; and denote by $\hat{S}$ the sorted list of S in non-increasing order. We define the lexicographical order. If S_1 and S_2 are two sequences of integers, $S_1 \leq_{lex} S_2$ if $S_1 = S_2$ or there is an i such that $S_1(i) < S_2(i)$ and $S_1(j) = S_2(j)$ for all $j < i$.

2 Forward Degree Sequences

Definition 1. *Given a graph G and a permutation σ on the vertex set $V(G)$, the* forward degree *of vertex $v = \sigma(i)$, denoted d_v^σ, is its degree in the induced*

subgraph $G' = G[\{\sigma(i), \sigma(i+1), \cdots, \sigma(n)\}]$. *The* forward degree sequence *induced by* σ, *denoted* S_σ, *is the sequence* $(d^\sigma_{\sigma(1)}, \ldots, d^\sigma_{\sigma(n)})$.

Definition 2. *For any sequence* S *of* n *non-negative integers, define* P_S *to be the polynomial* $P_S(z) = \sum_{i=1}^{n} z^{S(i)} = \sum_{k=0}^{\infty} N_S(k) z^k$. *Given a graph* $G = (V, E)$ *and a permutation* σ *of its vertices, the* forward degree sequence polynomial *induced by* σ *is* $P_\sigma(z) := P_{S_\sigma}(z) = \sum_{v \in V} z^{d^\sigma_v} = \sum_{k=0}^{\infty} N_{S_\sigma}(k) z^k$.

Any two forward degree sequences π and σ of G have the same length (the number of vertices in G) and the same sum (the number of edges in G). It follows that $P_\pi(1) = P_\sigma(1)$ and the derivatives $P'_\pi(1) = P'_\sigma(1)$. So we have

Proposition 1 *For any two forward degree sequences* π *and* σ *of* G, *the polynomial* $P_\pi - P_\sigma$ *is a multiple of* $(z-1)^2$.

The forward degree sequences of G carrie a lot of graph theoretic informations about G. We start as a warm up by presenting a lemma where the forward degree sequences are related to the structure of the graph.

Suppose a graph G has k connected components and σ is any permutation on vertices, then $N_{S_\sigma}(0) \geq k$ since the last vertex from each component in the order always has forward degree 0. Actually we can construct an ordering where the forward degree sequence contains exactly k zeros. The following lemma, which is a slightly stronger statement, can be easily proved.

Lemma 1. *If* G *is a connected graph and* v *is any vertex in* G, *then there is a permutation* σ *on the vertices of* G *such that* $N_{S_\sigma}(0) = 1$ *and the last vertex in the permutation is* v.

Definition 3. *Let* $G = (V, E)$ *be a graph, we define* $\mathcal{S}_G$ *to be the set of all the forward degree sequences, i.e.,* $\mathcal{S}_G = \{S_\sigma : \sigma \text{ is a permutation of vertices in } G\}$. *Define* $\hat{\mathcal{S}}_G$ *to be the set* $\{\hat{S} : S \in \mathcal{S}_G\}$. *Define* $\mathcal{P}_G$ *to be the set of all the forward degree sequence polynomials.*

One can easily see that $\mathcal{S}_G$ contains the information of many properties of G, such as the maximum degree, the minimum degree, the number of connected components, the size of the largest clique, the size of the largest independent set, and the girth.

3 Balanced and Strongly Balanced Sequences

For any integers $n, m \geq 0$, let $\hat{\mathcal{S}}_{n,m}$ be the set of all non-increasing, non-negative integer sequences of length n and the sum of elements m. We define a relation on $\hat{\mathcal{S}}_{n,m}$: $S_1 \precsim S_2$ if $P_{S_1}(z) \leq P_{S_2}(z)$ for all $z \geq 1$. It is easy to check $\precsim$ is a partial order on the ordered sequences. If $S_1 \precsim S_2$, we also write $P_{S_1} \precsim P_{S_2}$. Thus we view $\precsim$ as a partial order on corresponding polynomials.

Let G be a graph with n vertices and m edges, we have the induced partial orders $(\hat{\mathcal{S}}_G, \precsim)$ and $(\mathcal{P}_G, \precsim)$. Moreover, for any two orderings σ and π of vertices,

we write $\sigma \precsim \pi$ and $S_\sigma \precsim S_\pi$ if $\hat{S}_\sigma \precsim \hat{S}_\pi$, this is equivalent to $P_\sigma \precsim P_\pi$. $\precsim$ is no longer a partial order on all the permutations or on all the forward degree sequences of G, but it is still transitive and reflexive. We have an equivalence relation $\sigma \sim \pi$ if $\sigma \precsim \pi$ and $\pi \precsim \sigma$. (Similarly for the forward degree sequences.)

Definition 4. *A graph G is called* balanced *if there exist a minimum element in $(\hat{\mathcal{S}}_G, \precsim)$. The class of all the balanced graphs is denoted by $\mathcal{B}$.*

For any permutation σ, we call σ and S_σ $\precsim$ minimal if $\hat{S}_\sigma$ is minimal. If there exists a permutation of vertices σ such that $\sigma \precsim \pi$ for any permutation π of vertices, we call σ a $\precsim$ *minimum ordering* of G, and S_σ a $\precsim$ *minimum forward degree sequence*.

Let S be a sequence in $\hat{\mathcal{S}}_{n,m}$. If a and b both appears in S and $a \geq b+2$, the concentration operation $(a, b) \to (a-1, b+1)$ is performed by changing one a and one b to an $a-1$ and a $b+1$, then sort the sequence in non-increasing order. We define $S_1 \precsim_S S_2$ if we can get S_1 from S_2 by 0 or more steps of concentration. In this case we also write $P_{S_1} \precsim_S P_{S_2}$. If σ and π are two permutations on the vertices of a graph G, we write $\sigma \precsim_S \pi$ and $S_\sigma \precsim_S S_\pi$ if $\hat{S}_\sigma \precsim_S \hat{S}_\pi$ (and $P_\sigma \precsim_S P_\pi$). It is easy to see $S_1 \precsim S_2$ if $S_1 \precsim_S S_2$; $\precsim$ are partial orders on $\hat{\mathcal{S}}_{n,m}$, $\hat{\mathcal{S}}_G$, and $\mathcal{P}_G$.

Definition 5. *A graph G is called* strongly balanced *if there exist a minimum element in $(\hat{\mathcal{S}}_G, \precsim_S)$. The class of all the strongly balanced graphs is denoted by $\mathcal{B}_S$.*

If G is strongly balanced, any permutation σ which $\hat{S}_\sigma$ is minimum is called a $\precsim_S$ *minimum ordering*, S_σ is called a $\precsim_S$ *minimum forward degree sequence*, or a *most strongly balanced forward degree sequence*; if $\hat{S}_\sigma$ is $\precsim_S$ minimal, we also call σ or S_σ $\precsim_S$ minimal.

As an example, we note that every tree has a most strongly balanced forward degree sequence $(1, 1, \cdots, 1, 0)$, which is $\precsim_S$ minimum even in the whole $\hat{\mathcal{S}}_{n,n-1}$. Similarly, every forest is in $\mathcal{B}_S$. (See Corollary 1 for a generalization of this fact.)

There are several characterizations of the relation $\precsim_S$. We give one of them below.

Proposition 2 *Let S_1 and S_2 be two sequences in $\hat{\mathcal{S}}_{n,m}$. $S_1 \precsim_S S_2$ if and only if $P_{S_2} - P_{S_1} = (z-1)^2 Q$ where Q is a polynomial in z with positive integer coefficients.*

Sketch of Proof. The *only if* part is trivial. The *if* part may be proved by induction on the sum of coefficients in Q. The key obeservation is that for any "segment" (a, b) in Q ($b \leq a$ and the coefficient of z^i in Q is positive for each $b \leq i \leq a$, while the coefficient of z^{a+1} and z^{b-1} are both 0), we could change S_2 to S' by $(a+2, b) \to (a+1, b+1)$ and $P_{S'} - P_{S_1} = (z-1)^2 Q'$, where $Q' = Q(z) - (z^b + \ldots + z^a)$ is positive. □

Given a graph G, let S^* be the lexicographically smallest sequence in $\hat{\mathcal{S}}_G$. When z is big enough, $P_{S^*}(z) < P_S(z)$ for any other $S \in \hat{\mathcal{S}}_G$. Therefore, if $G \in \mathcal{B}$ or $G \in \mathcal{B}_S$, we must have S^* as the minimum sequence. From S^* we define some interesting graph parameters.

Definition 6. *Let G be a graph and S^* be the smallest lexicographical sequence in $\hat{\mathcal{S}}_G$. Define $m(G) := S^*(1)$, i.e., the largest forward degree in S^*. Define $N_k(G) := N_{S^*}(k)$ to be the number of occurrences of k in S^*. And define $N(G) := N_{m(G)}(G) = N_{S^*}(m(G))$.*

Let G be a graph and G' be a subgraph of G, then $m(G') \leq m(G)$. Moreover, if v is a vertex in G with the minimum degree, and $G' = G - v$, then $m(G) = \max\{d_v, m(G')\}$. From these observations, we see that $m(G)$ is computable in polynomial time by the following algorithm: We start from the empty sequence; find a vertex v in G with the smallest degree; put v as the next element in our sequence and delete v from G. Iterate this until G is empty. Thus we get an ordering of the vertices and a forward degree sequence. We claim $m(G)$ is the largest number in the sequence.

If $G \in \mathcal{B}_S$, the most strongly balanced forward degree sequence must be S^*. In finding S^*, we are trying to minimize the number of occurrences of $m(G)$ in a forward degree sequence. On the other hand, it is easy to see that the most strongly balanced sequence must contain as few 0's as possible, so the number of 0's in the sequence is exactly the number of connected components in G. With these observations, we can show there are graphs not in $\mathcal{B}_S$ and not even in $\mathcal{B}$.

4 Graphs Inside $\mathcal{B}$ or $\mathcal{B}_S$

4.1 Closure Properties

From a graph in $\mathcal{B}_S$ (or $\mathcal{B}$), we may construct new graphs in the same class by adding a vertex in some manner. The proofs of the following two propositions are easy shifting arguments.

Proposition 3 *If $G = (V, E) \in \mathcal{B}$ and $G' = (V \cup \{v\}, E \cup \{vw | w \in V\})$, then $G' \in \mathcal{B}$. Moreover, if $G \in \mathcal{B}_S$, then $G' \in \mathcal{B}_S$. If σ is a $\precsim$ ($\precsim_S$) minimum ordering of the vertices of G, then $\sigma' = (\sigma, v)$ is $\precsim$ ($\precsim_S$) minimum for G'.*

Proposition 4 *If $G = (V, E) \in \mathcal{B}$, K is a clique in G, and $G' = (V \cup \{v\}, E \cup \{vw | w \in K\})$, then $G' \in \mathcal{B}$. Moreover, if $G \in \mathcal{B}_S$, then $G' \in \mathcal{B}_S$. If σ is a $\precsim$ ($\precsim_S$) minimum ordering of the vertices of G, then $\sigma' = (v, \sigma)$ is $\precsim$ ($\precsim_S$) minimum for G'.*

Proposition 4 immediately gives the fact that there is a most strongly balanced sequence for any chordal graph.

Corollary 1. *The family of chordal graphs is contained in $\mathcal{B}_S$. Let G be a chordal graph. All simplicial elimination orderings of G give the same multi-set of the forward degree sequences.*

Proposition 5 *If $G = (V, E) \in \mathcal{B}_S$, x and y are two connected vertices in G, and $G' = (V \cup \{v\}, E \cup \{vx, vy\})$, then $G' \in \mathcal{B}_S$. If σ is a $\precsim_S$ minimum ordering of the vertices of G, then $\sigma' = (v, \sigma)$ is $\precsim_S$ minimum for G'.*

Proof. Without loss of generality, we may assume G is connected. Therefore, the number 0 appears exactly once in the forward degree sequence induced by σ. Clearly $P_{\sigma'}(z) = P(z) + z^2$.

Consider any ordering π' of G'. Without loss of generality, x comes before y in π'. By deleting v in π' we get π, an ordering of G. $P_\pi(z) - P(z) = (z-1)^2 Q(z)$, where Q is a polynomial in z with positive integer coefficients. Let $a = d_x^\pi$ and $b = d_y^\pi$.

Based on the position of v, x, and y in π', we have 3 cases.

(a) If v comes before x and y, then $d_v^\sigma = 2$ and $d_w^\sigma = d_w^{\sigma'}$ for any w in G. So $P_{\pi'}(z) = P_\pi(z) + z^2$. $P_{\pi'}(z) - P_{\sigma'}(z) = P_\pi(z) - P(z) = (z-1)^2 Q(z)$.

(b) If v comes between x and y, $P_{\pi'}(z) = P_\pi(z) + z + z^{a+1} - z^a$. $P_{\pi'}(z) - P_{\sigma'}(z) = P_\pi(z) - P(z) + z - z^2 + z^{a+1} - z^a = (z-1)^2 Q(z) + (z-1)(z^a - z)$. If $a \neq 0$, $(P_{\pi'}(z) - P_{\sigma'}(z))/(z-1)^2$ is a polynomial with positive integer coefficients. Otherwise, x has forward degree 0 in π, yet x is not the last vertex. So π contains at least 2 vertices of forward degree 0. So the constant term of Q is positive, and $P_{\pi'}(z) - P_{\sigma'}(z) = (z-1)^2(Q(z) - 1)$, where all the coefficients of $Q(z) - 1$ are positive integers.

(c) If v comes after x and y, $P_{\pi'}(z) = P_\pi(z) + 1 + z^{a+1} - z^a + z^{b+1} - z^b$. $P_{\pi'}(z) - P_{\sigma'}(z) = (z-1)^2 Q(z) + (z-1)(z^a - z + z^b - 1)$. The analysis is exactly the same as in the previous case.

In each case, $(P_{\pi'} - P_{\sigma'})/(z-1)^2$ has positive coefficients, by proposition 2, σ' is $\precsim_S$ minimum for G'. □

4.2 Graphs with Low Degrees

We denote by $\mathcal{D}_k$ the class of graphs with all degrees at most k: $\mathcal{D}_k = \{G : \Delta(G) \leq k\}$. Any graph $\mathcal{D}_2$ is a disjoint union of paths and cycles. By the propositions in Section 4.1, $\mathcal{D}_2 \subset \mathcal{B}_S$. There are examples shown that $\mathcal{D}_3 \not\subseteq \mathcal{B}_S$ and $\mathcal{D}_4 \not\subseteq \mathcal{B}$. In the rest of this section we prove that $\mathcal{D}_3 \subset \mathcal{B}$.

Biconnectivity and Block Structures. Recall some definitions and facts about the biconnected graphs and blocks. A graph is called *biconnected* if it is connected, has at least 3 vertices and contains no cut point. A maximal connected subgraph that has no cut point is called *a block*. In the standard approach the blocks is a partition of the edges of a graph. Blocks form a cactus-like structure and the so called block-cutpoint graph is a tree. In our approach we use an essentially same yet slightly different decomposition. We are going to partition the vertices.

Definition 7. *Given a graph G. A maximal biconnected subgraph of G is called* a cluster. *A vertex which does not belong to any biconnected subgraph is called a* router. *A* room *is either a cluster or a router.*

Clearly, a router is a vertex that does not belong to any cycle. Either it is an isolated point, or every block containing it is a bridge. Let G be a graph in $\mathcal{D}_3$, let R_1 and R_2 be any two distinct rooms in G, it follows from the standard

properties of the block decomposition of a graph (See [1] and [7]) that R_1 and R_2 do not share any common vertex, and here is at most one edge in G between the vertex set of R_1 and the vertex set of R_2. Thus, for graphs in $\mathcal{D}_3$, the rooms is a partition of vertices. we can define a super graph as

Definition 8. *Let $G \in \mathcal{D}_3$ with rooms $R_1, \cdots, R_k$, the* building map *G^R is the graph with vertex set $\{R_1, \cdots, R_k\}$ and R_iR_j is an edge in G^S if and only if there is an edge between R_i and R_j in G.*

For any $G \in \mathcal{D}_3$, the graph G^S is a forest. The number of connected components in G^S is the number of connected components in G.

Biconnected Graph in $\mathcal{D}_3$. If G is a connected graph in $\mathcal{D}_d$, and there is at least one vertex with degree smaller than d, then we can always find a forward degree sequence without any forward degree d. Indeed, we can pick any vertex of degree less than d, and notice that in the new graph there is at least one vertex of degree less than d in each of its connected component. Thus we have,

Proposition 6 *For any connected graph G with maximum degree d and minimum degree less than d, we have $m(G) < d$.*

Corollary 2. *For any d-regular graph G, $m(G) = d$ and $N(G)$ equals the number of connected components in G.*

For the biconnected graphs in $\mathcal{D}_3$, we have a similar yet stronger statement. In the proof of the next Lemma and in the rest of this section, we call a forward degree sequence (an ordering of vertices) *x-y-good* if it starts with x, ends with y, avoids any 3, and contains only one 0.

Lemma 2. *Suppose $G = (V, E)$ is a biconnected graph in $\mathcal{D}_3$. For any $v_s \in V$ such that $d^G_{v_s} = 2$ and any $v_f \in V - v_s$, there exists an ordering of vertices σ such that it starts with v_s, ends with v_f, and the forward degree sequence obtained by σ on G satisfies $N_{S_\sigma}(3) = 0$ and $N_{S_\sigma}(0) = 1$, i.e., it avoids 3 and contains only one 0.*

Proof. The proof is by induction on n, the number of vertices in G. The base case, when $n = 3$, is trivial. For n greater than 3, $G' = G - v_s$ is a connected graph in $\mathcal{D}_3$ with $n - 1$ vertices. Suppose x and y are the two neighbors of v_s, in G' we have $1 \le d^{G'}_x, d^{G'}_y \le 2$. We discuss two possible cases.

1. G' is biconnected.

In this case, $d^{G'}_x = d^{G'}_y = 2$ and at least one of them, say x, is not v_f. $\sigma = (v_s, \sigma')$ is v_s-v_f-good in G , where σ' is x-v_f-good in G'.

2. G' is not biconnected.

In this case, there should exist v_c which is a cut point of G'. Since the degree of v_s in G is 2 and G is biconnected, $G' - v_c$ has exactly two connected components, say, $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$, and v_s has neighbor in both components. We assume $x \in V_1$ and $y \in V_2$.

Based on H_1 and H_2, we define two new graphs. $G_1 = G[V_1 \cup \{v_c, v_s\}] + v_s v_c$, $G_2 = G[V_2 \cup \{v_c, v_s\}] + v_s v_c$. It is easy to show that they are biconnected graphs in $\mathcal{D}_3$ with at least three and less than n vertices, and v_s is a degree 2 vertex in both G_1 and G_2.

Now we construct a v_s-v_f-good ordering in G. Without loss of generality, v_f is in G_1. By induction, there is a v_s-v_f-good ordering (v_s, σ_1, v_f) in G_1, and a v_s-v_c-good ordering (v_s, σ_2, v_c) in G_2. Let $\sigma = (v_s, \sigma_2, \sigma_1, v_f)$. It is routine to check that σ is v_s-v_f-good in G. □

We prove that a v_s-v_f-good forward degree sequence, for any v_s and v_f with the degree of v_s being 2, is a most strongly balanced sequence for the graph.

Corollary 3. *Any biconnected graph with maximum degree* 3 *and minimum degree less than* 3 *belongs to the class* $\mathcal{B}_S$.

Proof. Let σ be a v_s-v_f-good ordering for any v_s and v_f with the degree of v_s being 2. Consider any other ordering π and the maximum number d appearing in S_π.

Since the graph is biconnected, $d > 1$. If $d = 2$, then $P_\pi(z) - P_\sigma(z) = (z-1)^2 c$ for some integer c. Since S_σ contains only one 0, the constant term of $P_\pi(z) - P_\sigma(z)$ is non-negative. If $d = 3$, then $P_\pi(z) - P_\sigma(z) = (z-1)^2(az+b)$ for some integer $a > 0$ and b. Again, since S_σ contains only one 0, the constant term of $P_\pi(z) - P_\sigma(z)$ is non-negative. In either case, $\sigma \precsim_S \pi$ by Proposition 2. □

In the proof of Lemma 2, we actually outlined an algorithm to find a good sequence. The complexity of the algorithm is easily analyzed. We have

Proposition 7 *Given any biconnected graph with* n *vertices, and with maximum degree* 3 *and minimum degree less than* 3*, the most strongly balanced (ordered) forward degree sequence is computable in* $O(n^3)$ *time.*

$\mathcal{B}$ Contains $\mathcal{D}_3$.

Lemma 3. *Any connected graph* G *with maximum degree* 3 *and minimum degree less than* 3 *belongs to the class* $\mathcal{B}$*. Moreover, the* $\precsim$ *minimum forward degree sequence is computable in polynomial time.*

Proof. We call a vertex *loose* if its degree is less than 3. We analyze the clusters in G and the building map G^S. We call a cluster R *bad* if it is a leaf in G^S and all its vertices has degree 3 in G. Let b be the number of bad clusters.

If we want to avoid the forward degree 3, each of the bad clusters contributes at least one 0. Thus, in any forward degree sequence of G that avoids 3, the number of 0's is at least $\max\{1, b\}$.

On the other hand, we can always find a forward degree sequence of G that avoids 3 and contains $\max\{1, b\}$ 0's: Pick any loose vertex x and find its room R_r. We view G^S as a rooted tree with root R_r. For any leaf L which is not a bad cluster, by Lemma 2, we can eliminate it without producing any 3 or 0, or changing the number of bad clusters. We repeat this until all the leaves are bad. Now, we find an order from the root down to the leaves. We start from x. For

any non-leaf room R, we start from a loose vertex, and eliminate its vertices according to the ordering provided by Lemma 2 which ends in any of its ports to R's children of R. Thus we do not have any forward degree 3 or 0, and created a loose vertex for each of R's children. Finally we have d leaves in G^S, accordingly d connected components in G each has a loose vertex. By Lemma 2, we finish by d orderings containing one 0 each.

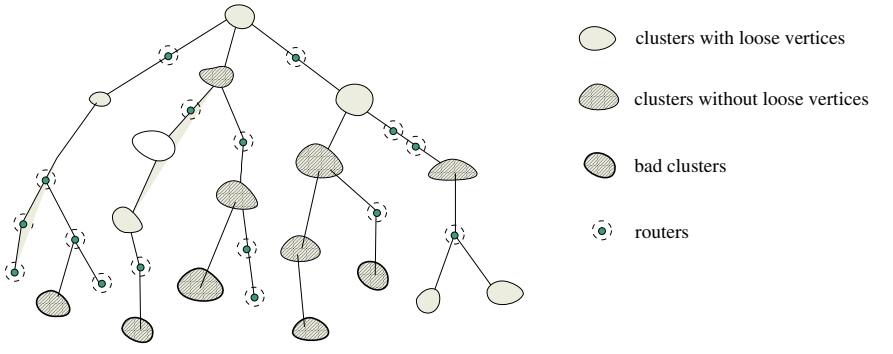


Fig. 1. The good leaves are treated bottom up, then the non-leaf nodes are treated top down.

Now we finish the proof of Lemma 3 by showing that any such sequence σ is $\precsim$ minimum for G. We prove this by induction on the size of G. The base case is trivial. Assume for any graph with less vertices there is a $\precsim$ minimum ordering that has as few 0's as possible under the condition that 3 does not appear.

Consider any other ordering π. We consider two cases.

1. π avoids the forward degree 3. Then, $P_\pi(z) - P_\sigma(z) = (z-1)^2 c$ for some integer c. Because σ has the fewest number of 0's among all forward degree sequence without 3, $c \geq 0$ and hence $\sigma \precsim \pi$.

2. 3 appears in π as the forward degree for some v. We may assume it appears as the beginning and, by the inductive hypothesis, $\pi = (v, \pi')$, where π' is the $\precsim$ minimum sequence of $G - v$. So, 3 appears exactly once in S_π. $P_\pi(z) - P_\sigma(z) = (z-1)^2(z-c)$ for some integer c.

Our goal is to show that $c \leq 1$, equivalently, $N_{S_{\pi'}}(0) \geq N_{S_\sigma}(0) - 1$. If there is no bad clusters in G, then $N_{S_\sigma}(0) = 1$ thus $N_{S_{\pi'}}(0) \geq N_{S_\sigma}(0) - 1$. Otherwise, there are $b \geq 1$ bad clusters. $G - v$ contains at least $b - 1$ bad clusters, $N_{S_{\pi'}}(0) \geq b - 1 = N_{S_\sigma}(0) - 1$.

The procedure we outlined in this proof gives a best forward degree sequence in polynomial time, provided the algorithm in Proposition 7 as a sub-routine. □

Now we are ready to prove our main result in this section.

Theorem 1 *Any graph with all degrees at most* 3 *has* $\precsim$ *minimum forward degree sequence; and there is a polynomial time algorithm computes the* $\precsim$ *minimum forward degree sequence. Especially, the class of* 3*-regular graphs is contained in* $\mathcal{B}$.

Proof. We may assume the graph G is connected. If there is a vertex with degree less than 3, the statement is true by Lemma 3. Otherwise, G is 3-regular, with vertices v_1, v_2, $\cdots$, v_n. Let $G_i = G - v_i$, each connected component of G_i has a vertex of degree less than 3. Therefore, by Lemma 3, there is a $\precsim$ minimum forward degree sequence σ_i' for G_i. Let $\sigma_i = (v_i, \sigma_i')$, then $P_{\sigma_i}(z) = z^3 + P_{\sigma_i'}(z)$.

For any ordering (v_i, σ'), $(v_i, \sigma_i') \precsim (v_i, \sigma')$. So we only need to show that there is a $\precsim$ minimum polynomial among $P_{\sigma_1}, \cdots, P_{\sigma_n}$.

Actually $\precsim$ is a linear order on the set of $\{P_{\sigma_i} : 1 \leq i \leq n\}$. For any i and j, being the $\precsim$ minimum sequences, $S_{\sigma_i'}$ and $S_{\sigma_j'}$ do not contain any forward degree larger than 2. So,

$$P_{\sigma_i'}(z) - P_{\sigma_j'}(z) = P_{\sigma_i}(z) - P_{\sigma_j}(z) = (z-1)^2 c$$

for some integer c. That is, they are $\precsim$ ($\precsim_S$) comparable. □

5 Conclusion

In this paper we have defined and studied forward degree sequences and their associated polynomials. In particular, the properties of (strongly) balanced forward degree sequences are investigated. Our proof shows that any chordal graph has a strongly balanced forward degree sequence and any graph with all degrees at most 3 has a balanced forward degree sequence. Moreover, these (strongly) balanced forward degree sequences can be computed in polynomial time. Our results might bring a new clue for graph problems related to vertex ordering, such as graph isomorphism, because a (strongly) balance forward degree sequence is an optimal vertex ordering. Our work could be extended by finding more classes of graphs which are (strongly) balanced. Also, it is still open that whether some of the graph invariants derived naturally from forward degree sequence, such as $N(G)$, are polynomial time computable.

References

1. B. Bollobás, *Modern Graph Theory*, Springer-Verlag, New York (1998).
2. P. Erdos and T. Gallai, *Graphs with Prescribed Degrees of Vertices*, Mat. Lapok. 11, pp264-274, (1960).
3. S. Hakimi, *On the Realizability of a Set of Integers as Degrees of the Vertices of a Graph*, SIAM J. Appl. Math. 10, pp496-506, (1962).
4. V. Havel, *A Remark on the Existence of Finite Graphs*, Casopis Pest. Mat. 80, pp477-480, (1955).
5. C. M. Hoffmann, *Group-theoretic algorithms and graph isomorphism*, volume 136 of Lecture Notes in Computer Science, Springer-Verlag Inc., New York, (1982).
6. E. M. Luks, *Isomorphism of bounded valence can be tested in polynomial time*, Proc. of the 21st Annual Symposium on Foundations of Computing, pp42-49. IEEE, (1980).
7. D. B. West, *Introduction to Graph Theory*, Prentice Hall, New Jersey (1996).

Conditionally Critical Indecomposable Graphs

Chandan K. Dubey[1], Shashank K. Mehta[1,⋆], and Jitender S. Deogun[2]

[1] Indian Institute of Technology, Kanpur – 208016, India
{cdubey,skmehta}@cse.iitk.ac.in
[2] University of Nebraska-Lincoln, Lincoln, NE 68588-0115, USA
deogun@cse.unl.edu

Abstract. Let X be a subset of vertices of an undirected graph $G = (V, E)$. G is X-critical if it is indecomposable and its induced subgraph on X vertices is also indecomposable, but all induced subgraphs on $V - \{w\}$ are decomposable for all $w \in V - X$. We present two results in this paper. The first result states that if G is X-critical, then for every $w \in V - \{x\}$, $G[V - \{w\}]$ has a unique non-trivial module and its cardinality is either 2 or $|V| - 2$. The second result is that the vertices of $V - X$ can be paired up as $(a_1, b_1), \ldots, (a_k, b_k)$ such that induced subgraphs on subset $V - \{a_{j_1}, b_{j_1}, \ldots, a_{j_s}, b_{j_s}\}$ are also X-critical for any collection of pairs $\{(a_{j_1}, b_{j_1}), \ldots, (a_{j_s}, b_{j_s})\}$.

1 Introduction

A *module (or interval)* of an undirected graph $G = (V, E)$ is a subset of vertices, $M \subseteq V$ such that for any $a, b \in M$ and $c \in V - M$ edge $(a, c) \in E$ if and only if edge $(b, c) \in E$. Singleton sets and the entire vertex set are modules vacuously, and therefore are called *trivial modules*. A graph is said to be indecomposable when it does not contain a non-trivial module.

Some recent works have studied how indecomposability is inherited by induced subgraphs. Here we mention two papers which were the prime motivation for the current work. Schmerl and Trotter [1] studied critically indecomposable graphs. These graphs are such that any induced subgraph on one fewer vertex is decomposable. The study showed that the family of bipartite graphs on vertex set $\{u_1, \ldots, u_m\} \cup \{v_1, \ldots, v_m\}$ where each u_j is adjacent to v_k for $k \geq j$ are critically indecomposable. The only other graphs which are also critically indecomposable are the complement of these graphs.

Cournier and Ille [2] have studied criticality from a different perspective. They define minimality by requiring that every proper subgraph of indecomposable graph G must be decomposable if it properly contains an indecomposable subgraph of $G[X]$.

Pierre Ille [3] has shown that if G is indecomposable and has an induced indecomposable subgraph on vertex set X, then there is a set Y containing

⋆ Partly supported by Ministry of Human Resource Development, Government of India under grant no. MHRD/CD/20030320

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 690–700, 2005.

X and $|Y| = |V| - 2$ such that $G[Y]$ is also indecomposable, subject to some minimal conditions on the sizes of X and $V - X$.

In this work we generalize the notion of critical indecomposability of [1] inspired by [3]. Let X be a subset of vertices of G. Then G is *X-critically indecomposable* if G and its induced subgraph on X are indecomposable but every induced subgraph on $V - \{w\}$, $w \in V - X$, is decomposable. For simplicity we will refer to this property by *X-ci*. The critically indecomposable graphs defined by Schmerl and Trotter [1] are $\emptyset$-ci. We also show that the vertices of $V - X$ in an X-ci graph can be paired up such that the reduced graph after the deletion of any collection of these pairs is also X-critical.

2 Critical Indecomposability and Related Notions

Notation In this paper G will denote a graph with vertex set V and edge set E. If Y is a subset of V, then $G[Y]$ will denote the induced subgraph of G on vertex set Y. Symmetric 0-1 matrix e will represent adjacency. So $e_{uv} = 1$ if and only if edge (u, v) belongs to E. Throughout the paper we will deal with *undirected graphs* although the proofs can be easily adapted for directed graphs.

A property of modules trivially deducible from the definition is as follows.

Proposition 1. *(i) M is a module of G and $Y \subseteq V$. Then $M \cap Y$ is a module of $G[Y]$.*
(ii) If M_1 and M_2 are modules of G such that $M_1 \cap M_2$ is non-empty then $M_1 \cup M_2$ is also a module of G.

We reproduce here a basic result from [1, 4] which is used in the current work.

Theorem 1. *[1, 4] Let $G = (V, E)$ be an indecomposable graph with an indecomposable subgraph $G[X]$ s.t. $3 \leq |X| \leq |V| - 2$. Then there is an indecomposable subgraph $G[Y]$ s.t. $X \subset Y \subseteq V$ and $|Y| = |X| + 2$.*

Definition 1. *$G = (V, E)$ is said to be* marginally decomposable *if (i) there is only one module in the graph, and (ii) the size (vertex cardinality) of the module is either 2 or $|V| - 1$.*

Next we define a notion which is more stringent than X-criticality.

Definition 2. *Graph G has an indecomposable subgraph $G[X]$. Then G is said to be* X-stably-indecomposable *(in short X-si) if (i) G is indecomposable, and (ii) $G[V - \{w\}]$ is marginally decomposable for all $w \in V - X$.*

It is obvious that every X-si subgraph is X-ci. In the sequel we will show that the two concepts are equivalent.

Lemma 1. *Let Y be a subset of 4 or more vertices of G and $x \in Y$. Then*
(a) if $G[Y - \{x\}]$ is indecomposable and $G[Y]$ is decomposable, then $G[Y]$ is marginally decomposable.

(b) if $G[Y - \{x\}]$ is indecomposable and $G[Y]$ is marginally decomposable, then the module of $G[Y]$ is either $Y - \{x\}$ or $\{x, y\}$ for some $y \in Y - \{x\}$.
(c) $G[Y]$ is marginally decomposable with the module $Y - \{x\}$ or $\{x, y\}$ where $y \in Y - \{x\}$, then $G[Y - \{x\}]$ is indecomposable.

Proof. (a) Let M be a non-trivial module of $G[Y]$. $G[Y - \{x\}]$ is indecomposable so $M' = M \cap (Y - \{x\})$ must be a trivial module of $G[Y - \{x\}]$. Thus M' can be either $\{y\}$ or $Y - \{x\}$, for some $y \in Y - \{x\}$. Therefore M will be $\{x, y\}$ or $Y - \{x\}$, since it is a non-trivial module of $G[Y]$. Next we show that at most one M is possible.

Let M_1 and M_2 are modules of $G[Y]$. There are two cases to be considered: (i) $M_1 = \{x, y_1\}, M_2 = \{x, y_2\}$ and (ii) $M_1 = \{x, y\}, M_2 = Y - \{x\}$. In case (i) $\{y_1, y_2\}$ is a module of $G[Y - \{x\}]$ and in case (ii) $Y - \{x, y\}$ is a module of $G[Y - \{x\}]$. In each case the module is non-trivial so it contradicts the fact that $G[Y - \{x\}]$ is indecomposable.

(b) This claim is established in part (a).

(c) Consider the case where $\{x, y\}$ is the module of $G[Y]$. Assume that M is a module of $G[Y - \{x\}]$. If $y \in M$ then $M \cup \{x\}$ is a module of $G[Y]$. Uniqueness requires that $M \cup \{x\} = \{y, x\}$ thus $M = \{y\}$, i.e., M is trivial. If $y \notin M$, then M is also a module of $G[Y]$. In this case uniqueness requires that $M = \{x, y\}$ which is not possible since $y \notin M$.

Next consider the case of module $Y - \{x\}$. In this case x is either connected to all vertices of $Y - \{x\}$ or not connected to any. If M is a module of $G[Y - \{x\}]$, then it must also be a module of $G[Y]$. Thus $M = Y - \{x\}$, but this is a trivial module of $G[Y - \{x\}]$.

Lemma 2. *$G = (V, E)$ is X-ci with $|X| \geq 3$, and $G' = G[V - \{a, b\}]$ is indecomposable for some $a, b \in V - X$. Then $G[V - \{a, b\}]$ is also X-ci.*

Proof. Suppose G' is not X-ci. So there exists $c \in V - X - \{a, b\}$ such that $G[V - \{a, b, c\}]$ is also indecomposable. Since $|V - \{a, b, c\}| \geq |X| \geq 3$, we can use Theorem 1 to deduce that there are u, v in $\{a, b, c\}$ such that $G[V - \{a, b, c\} \cup \{u, v\}]$ is indecomposable. This graph is $G'' = G[V - \{w\}]$ where w is one of a, b, c. On the contrary, by definition of X-ci, G'' is decomposable.

Definition 3. *If G is X-ci and $a, b \in V - X$ such that $G[V - \{a, b\}]$ is also X-ci then the unordered pair (a, b) will be called a locked pair of G. Indeed it depends on X.*

Lemma 3. *Let $G = (V, E)$ is X-ci with $V - X$ non-empty and $|X| \geq 3$. Then G has a locked pair.*

Proof. Consider the indecomposable subgraph $G[X]$. From Theorem 1 we know that there is an indecomposable subgraph $G[Y]$ such that $X \subset Y$ and $|Y| = |X| + 2$. Repeating the argument we find that there is an indecomposable subgraph $G[V']$ such that $X \subseteq V'$ and $1 \leq |V - V'| \leq 2$. But $|V - V'|$ cannot be 1 since G is X-ci and V' contains X. Suppose $V' = V - \{a, b\}$. From lemma 2 we conclude that (a, b) is a locked pair in G.

An X-ci subgraph cannot have vertex cardinality equal to $|X|+1$ because of criticality condition. Combining this fact with lemma 3 leads to the following corollary.

Corollary 1. *$G=(V,E)$ is an X-ci graph with $|X|\geq 3$ then $|V-X|=2k$ for some $k\geq 0$.*

Consider an indecomposable subgraph $G[Y]$. We can partition the vertices of $V-Y$ as follows. Let $x\in V-Y$. If $G[Y\cup\{x\}]$ is indecomposable then x will be said to belong to class $G[Y]$-Ext (extension). In case $G[Y\cup\{x\}]$ is decomposable then from lemma 1 there are two possibilities: Y is a module or $\{x,y\}$ is module in $G[Y\cup\{x\}]$. In the former case x belongs to $G[Y]$-Dom (dominator) and in the latter case it belongs to $G[Y]$-Equi (equivalent). This terminology is adopted from [5], [6].

Let G be an X-ci graph and (a,b) be a locked pair. $G[V-\{a\}]$ and $G[V-\{b\}]$ are decomposable so neither vertex can belong to $G[V-\{a,b\}]$-Ext. Further, both vertices cannot belong to $G[V-\{a,b\}]$-Dom because that would imply that $V-\{a,b\}$ is a module of G which is absurd since G is indecomposable.

Proposition 2. *Let (a,b) be a locked-pair in an X-ci graph $G=(V,E)$. Then either (i) both a and b are in class $G[V-\{a,b\}]$-Equi or (ii) one each is in $G[V-\{a,b\}]$-Equi and $G[V-\{a,b\}]$-Dom.*

3 Critical Is Stable

The main goal of this section is to prove that X-ci and X-si are equivalent. Every X-si is trivially X-ci so we only need to prove that X-ci implies X-si property. We shall establish this result by induction. Suppose G is X-ci and (a,b) is a locked pair. In Lemma 5 and 6 we will show that if all X-ci subgraphs of $G[V-\{a,b\}]$ are also X-si, then G is X-si. We begin with a supporting result.

Lemma 4. *$G=(V,E)$ is X-ci and $G[V-\{a,b\}]$ is X-si for some $a,b\in V-X$. If $a\in G[V-\{a,b\}]$-Dom and $b\in G[V-\{a,b\}]$-Equi with $\{b,p\}$ as the unique module of $G[V-\{a\}]$ and if $p\notin X$, then $G[V-\{p\}]$ is marginally decomposable and its module is $\{a,u\}$ where u is some vertex in $V-\{a,b,p\}$.*

Proof. By definition $a\in G[V-\{a,b\}]$-Dom means $V-\{a,b\}$ is a module of $G[V-\{b\}]$. Thus $e_{ax'}=e_{ax''}$ for any $x',x''\in V-\{a,b\}$. But $V-\{a,b\}$ cannot be a module of the entire graph G because G is indecomposable, therefore we must have $e_{ax}\neq e_{ab}$ for $x\in V-\{a,b\}$.

Let M be a non-trivial module of $G[V-\{p\}]$. Then due to stable indecomposability of $G[V-\{a,b\}]$ and p not in X, $|M\cap(V-\{a,b,p\})|$ must be 0, 1, 2, $|V|-4$, or $|V|-3$.

If a and b both belong to M, then $M\cup\{p\}$ will become a non-trivial module of G, because $\{b,p\}$ is a module of $G[V-\{a\}]$, i.e., b and p have same connectivity with $V-\{a,b,p\}$. If neither of the two vertices belong to M, then it is also a non-trivial module of G. $b\in M$ but $a\notin M$ is not possible since $e_{ax}\neq e_{ab}$.

At last suppose $a \in M$ but $b \notin M$. Consider $M - \{a\}$. It is in $V - \{a, b\}$ so a has same connectivity with entire $M - \{a\}$. Besides, $\{b, p\}$ being a module of $G[V - \{a\}]$, all elements of $M - \{a\}$ have same connectivity with p because it is so with b. It implies that $M - \{a\}$ is a module of G, contradicting the fact that G is indecomposable. Therefore such case also cannot occur if $M - \{a\}$ is a non-trivial module. These considerations imply that the only possible modules of $G[V - \{p\}]$ are of the form $\{a, u\}$, where $u \in V - \{a, b, p\}$

To complete the proof we will show that $G[V - \{p\}]$ cannot have more than one non-trivial module. Let $\{a, u_1\}$ and $\{a, u_2\}$ be two of its modules. Using the fact that p and b are similarly connected to all vertices $V - \{a, b, p\}$ we find that $\{u_1, u_2\}$ is a module of $G[V - \{a, b\}]$, which is not possible because it is indecomposable.

Corollary 2. *$G = (V, E)$ is X-ci and $G[V - \{a, b\}]$ is X-si for some $a, b \in V - X$. If $a \in G[V - \{a, b\}]$-Dom and $b \in G[V - \{a, b\}]$-Equi with $\{b, p\}$ as the unique module of $G[V - \{a\}]$ and $p \notin X$, then the only module of $G[V - \{a, b, p\}]$ is $V - \{a, b, p, u\}$ where u is some vertex of $V - \{a, b, p\}$.*

Proof. From the lemma we know that the modules of $G[V - \{p\}]$ are of the form $\{a, u\}$ for some $u \in V - \{a, b, p\}$. Therefore u, like a, is either adjacent to all vertices of $V - \{a, b, p, u\}$ or not adjacent to all of $V - \{a, b, p, u\}$.

Lemma 5. *$G = (V, E)$ is X-ci with $|X| \geq 3$ and $|V| > 6$. If (a, b) is a locked pair with $a \in G[V - \{a, b\}]$-Dom and $b \in G[V - \{a, b\}]$-Equi. If every X-ci subgraph of $G[V - \{a, b\}]$, including itself, is X-si, then G is X-si.*

Proof. Let $p \in V - \{a, b\}$ such that $\{b, p\}$ is the unique module of marginally decomposable graph $G[V - \{a\}]$.

To prove the claim we have to show that $G[V - \{w\}]$ is marginally decomposable for all $w \in V - X$. We partition the cases into three categories.

(I) Cases of $w = a$ and $w = b$:
G being X-ci, $G[V - \{a\}]$ and $G[V - \{b\}]$ are decomposable. Since $G[V - \{a, b\}]$ is indecomposable, from lemma 1(a) we conclude that $G[V - \{a\}]$ and $G[V - \{b\}]$ are both marginally indecomposable.

(II) Case of $w = p$ when $p \notin X$:
Observe that $V - \{a, b\}$ contains X and graph $G[V - \{a, p\}]$ is isomorphic to $G[V - \{a, b\}]$ so it is also X-ci. G is X-ci so from lemma 1(a) we deduce that $G[V - \{p\}]$ is also marginally decomposable.

(III) Case of $w \in V - \{a, b, p\}$:
As a is given to be $G[V - \{a, b\}]$-Dom, it is either adjacent to all vertices of $V - \{a, b\}$ or not adjacent to all vertices of $V - \{a, b\}$. The reasoning remains same for both the cases. Therefore for simplification we shall assume that a is adjacent to all vertices of $V - \{a, b\}$. Since G is indecomposable, a and b must not be adjacent otherwise $G[V - \{a\}]$ will be a module of G.

Let M be a module of $G[V - \{w\}]$. Since $G[V - \{a, b\}]$ is X-si, $|M \cap (V - \{a, b, w\})|$ can be 0,1,2,$|V| - 4$ or $|V| - 3$. Further, a belongs to $G[V - \{a, b\}]$-Dom so M cannot contain b without containing a. Subject to these conditions we

consider all the possible values of M. We show some values of M are impossible because either it leads to structural inconsistency, or it requires $G[V - \{a, b, w\}]$ to have more than one non-trivial modules, which is not possible for a marginally decomposable graph.

For the rest of the cases it will be shown that M', the module of $G[V - \{a, b, w\}]$, is uniquely determined by M. For each possible M the corresponding M' will be different. This will imply that $G[V - \{w\}]$ has unique module for a given w since $G[V - \{a, b, w\}]$ has a unique module. Due to explicit construction of all possible M we will be able to verify that every feasible M has cardinality 2 or $|V| - 2$. In the following cases v, u will denote some vertices in $V - \{a, b, w\}$. In the following we take a group of values for M and describe its consequence.
(i) $\{a, b\}, \{a, p\}, \{a, b, p\}$: Then $V - \{a, b, p, w\}$ is a module of $G[V - \{a, b, w\}]$.
(ii) $\{a, v\}, \{a, v, u\}$: Since $e_{ab} = 0$, $e_{vb} = 0$. But $e_{ap} = 1$ so $e_{vp} = 1$. Since $\{b, p\}$ is a module of $G[V - \{a\}]$, $e_{vp} = 1$ implies that $e_{vb} = 1$. Hence contradiction.
(iii) $\{v, u\}$: Then $\{v, u\}$ is also a module of $G[V - \{a, b, w\}]$.
(iv) $\{a, b, v, u\}$: Not possible because $\{v, u\}$ and $\{v, u, p\}$ are both modules of $G[V - \{a, b, w\}]$. Here $\{v, u, p\}$ is a non-trivial module since $|V| > 6$.
(v) $\{p, v\}$: Then $\{p, v\}$ is also a module of $G[V - \{a, b, w\}]$.
(vi) $\{a, p, v\}, \{a, b, v\}, \{a, b, v, p\}$: Not possible since $\{v, p\}$ and $V - \{a, b, w, v, p\}$ are both modules of $G[V - \{a, b, w\}]$. Here again the second is a non-trivial module since $|V| > 6$.
(vii) $V - \{a, b, v, w\}, V - \{b, v, w\}$: not possible since $V - \{a, b, v, w\}$ and $V - \{a, b, v, w, p\}$ are modules of $G[V - \{a, b, w\}]$. Module $V - \{a, b, v, w, p\}$ is non-trivial as shown in case (vi).
(viii) $V - \{v, w\}$: here $V - \{a, b, v, w\}$ is a module of $G[V - \{a, b, w\}]$.
(ix) $V - \{b, w\}, V - \{p, w\}, V - \{a, b, w\}, V - \{a, b, p, w\}$: Then $V - \{a, b, p, w\}$ is the module of $G[V - \{a, b, w\}]$
(x) $V - \{b, p, w\}$: Let $v \in V - \{a, b, p, w\}$. Since $e_{ab} = 0$, e_{vb} is also 0. But $e_{ap} = 1$ so $e_{vp} = 1$. This contradicts the fact that $\{b, p\}$ is a module of $G[V - \{a\}]$.

To summarize, we must have module M' of $G[V - \{a, b, w\}]$ whenever $G[V - \{w\}]$ has module M:
(i) $M' = V - \{a, b, p, w\}$: for $M = \{a, b\}, \{a, p\}, \{a, b, p\}, V - \{a, b, p, w\}, V - \{p, w\}, V - \{a, b, w\}, V - \{b, w\}$,
(ii) $M' = V - \{a, b, v, w\}$: for $M = V - \{v, w\}$,
(iii) $M' = \{v, u\}$: for $M = \{v, u\}$,
(iv) $M' = \{p, v\}$: for $M = \{p, v\}$.

Next we will show that $M' = V - \{a, b, p, w\}$ is impossible so the first case cannot occur.

If $V - \{a, b, p, w\}$ is a module of $G[V - \{a, b, w\}]$, then from Lemma 1(c) $G[V - \{a, b, p, w\}]$ is indecomposable. Observe that $p \notin X$ otherwise $X - \{p\}$ would be a module of $G[X]$, which is indecomposable. By choice, $w \notin X$ so $X \subseteq V - \{a, b, p, w\}$. From Lemma 2, $G[V - \{a, b, p, w\}]$ is X-ci, i.e., $\{p, w\}$ is a locked pair of $G[V - \{a, b\}]$. Since $G[V - \{a, b, p, w\}]$ is a subgraph of $G[V - \{a, b\}]$, it is X-si. Since p is in $G[V - \{a, b, p, w\}]$-Dom, w belongs to $G[V - \{a, b, p, w\}]$-

Equi. Therefore $G[V - \{a, b, p\}]$ has module $\{w, v\}$ for some v. From Corollary 2 only possible module of $G[V - \{a, b, p\}]$ is $V - \{a, b, p, u\}$ for some u. This is not possible because $|V - \{a, b, p, u\}| > 2$ so $V - \{a, b, p, u\} \neq \{w, v\}$.

This leaves only three possibilities for the modules of $G[V - \{w\}]$: $M = V - \{v, w\}$, $M = \{v, u\}$ and $M = \{p, v\}$. But in each case different module for $G[V - \{a, b, w\}]$ must exist. Since $G[V - \{a, b\}]$ is X-si, $G[V - \{a, b, w\}]$ can have only one module. This forces $G[V - \{u\}]$ to have only one of the three possible modules for a fixed w. Besides, in each case the module of $G[V - \{w\}]$ has cardinality of 2 or $|V| - 2$. Thus $G[V - \{w\}]$ is marginally decomposable.

Table 1 shows the possible modules M of $G[V-\{w\}]$ and the implied modules M' of $G[V - \{a, b, w\}]$ for all $w \in V$.

Table 1. Case of $a \in G[V - \{a, b\}]$-Dom, $b \in G[V - \{a, b\}]$-Equi

w	M	M′	Remark
a	$\{b, p\}$		M' can't exist as $w \notin V - \{a, b\}$
b	$V - \{a, b\}$		M' can't exist as $w \notin V - \{a, b\}$
p			$\{p, b\}$ is the module of $V - \{a\}$
	$\{a, u\}$	$V - \{a, b, u, p\}$	u is some vertex of $V - \{a, b, p\}$
w			for any w in $V - \{a, b, p\}$
	$V - \{v, w\}$	$V - \{a, b, v, w\}$	v is some vertex of $V - \{a, b, p, w\}$
	$\{v, u\}$	$\{v, u\}$	v, u are some vertices of $V - \{a, b, p, w\}$
	$\{v, p\}$	$\{v, p\}$	v is some vertex of $V - \{a, b, p, w\}$

In the following lemma we consider the case when both, a and b, are in $G[V - \{a, b\}]$-Equi.

Lemma 6. *$G = (V, E)$ is X-ci with $|X| \geq 3$ and $|V| > 6$. If (a, b) is a locked pair with a and b both in $G[V - \{a, b\}]$-Equi. If $G[V - \{a, b\}]$ is X-si, then G is X-si.*

Proof. a and b are in $G[V - \{a, b\}]$-Equi so there exist $q, p \in V - \{a, b\}$ such that $\{a, q\}$ is the module of $G[V - \{b\}]$ and $\{b, p\}$ is the module of $G[V - \{a\}]$. Further p and q must be distinct because otherwise $\{a, b, p\}$ will be a module of G, which is an indecomposable graph.

We need to show that $G[V - \{w\}]$ is also marginally decomposable for all $w \in V - X$.

(I) Cases of $w = a$ and $w = b$: Same as case (I) of Lemma 5.

(II) Cases of $w = p$ when $p \notin X$ and $w = q$ when $q \notin X$:
$\{b, p\}$ is a module (to be precise, the only module) of $G[V - \{a\}]$ so $G[V - \{a, p\}]$ is isomorphic to $G[V - \{a, b\}]$. Therefore the former is also X-ci. From Lemma 1, $G[V - \{a, p\} \cup \{a\}] = G[V - \{p\}]$ is marginally decomposable with module $V - \{a, p\}$ or $\{a, u\}$ for some $u \in V - \{a, p\}$. We refine this claim as follows.

If $\{a, b\}]$ is a module of $G[V - \{p\}]$, then $\{p, q\}$ is a module of $G[V - \{a, b\}]$ which contradicts the fact that $G[V - \{a, b\}]$ is X-ci. Also $\{a, q\}$ cannot be a module of $G[V - \{p\}]$ because $e_{ab} \neq e_{qb}$. So the module of $G[V - \{p\}]$ is either $V - \{a, p\}$ or $\{a, u\}$ for some $u \in V - \{a, b, p, q\}$. Case of $w = q$ is similar.

Before proceeding to the last case where $w \in V - \{a, b, p, q\}$ we establish a useful result.

Claim $\{p, q\}$ is not the module of $G[V - \{a, b, w\}]$ for any $w \in V - \{a, b, p, q\}$.

Proof Assuming the contrary let $\{p, q\}$ be the module of $G[V - \{a, b, w\}]$ for some w. Both p and q cannot be inside X otherwise $\{p, q\}$ would be a module of $G[X]$. Let $p \in V - X$. From part (II) the only module of $G[V - \{p\}]$ is either $V - \{a, p\}$ or $\{a, u\}$ where $u \in V - \{a, b, p, q\}$. Therefore either $V - \{a, b, p, q\}$ or $\{q, u\}$ is a module of $G[V - \{a, b, p\}]$. Besides, it is the unique module since $G[V - \{a, b, p\}]$ is marginally decomposable.

Case: $\{q, u\}$ is the module in $G[V - \{a, b, p\}]$

If $u \neq w$, $\{p, q\}$ be the module of $G[V - \{a, b, w\}]$ and $\{q, u\}$ is the module in $G[V - \{a, b, p\}]$ implies $\{p, q, u\}$ is also a module of $G[V - \{a, b, w\}]$. But this is not possible because $G[V - \{a, b, w\}]$ is marginally decomposable. If $u = w$, then $\{p, q, u\}$ is a module (nontrivial because $|V| > 6$) of $G[V - \{a, b\}]$ which is also not possible as it is an indecomposable graph.

Case: $V - \{a, b, p, q\}$ is the module in $G[V - \{a, b, p\}]$

In this case $V - \{a, b, p, q, w\}$ should be the module of the marginally decomposable graph $G[V - \{a, b, w\}]$ (as $\{p, q\}$ is a module of $G[V - \{a, b, w\}]$) which is impossible because $|V| > 6$ so $V - \{a, b, p, q, w\}$ is non-trivial and $\{p, q\}$ is another module.

End of Proof of the Claim

Now we resume the proof of the lemma.

(III) Case of $w \in V - \{a, b, p, q\}$:

If M is a module of $G[V - \{w\}]$, then $M \cap (V - \{a, b, w\})$ must be either a trivial module or that of size 2 or $|V| - 4$ because $G[V - \{a, b, w\}]$ is marginally decomposable.

Without loss of generality let us assume that $e_{ab} = 0$. Therefore $e_{ap} = 1$ since otherwise $\{b, p\}$ will become a module in G, which is indecomposable. Similarly $e_{bq} = 1$. Since $\{a, q\}$ is a module in $G[V - \{b\}]$, $e_{pq} = 1$.

Observe that $e_{ab} \neq e_{qb}$ so M cannot contain a and q without containing b. Similarly it cannot contain b, p without containing a.

Under the constraints mentioned above, only following values of M are left to be considered. Here u, v are some vertices in $V - \{a, b, p, q\}$. We will exploit the symmetry between pair (a, q) and (b, p) to reduce the cases.

(i) $\{a, b\}, \{a, p\}, \{a, b, p\}, \{p, q\}, \{a, b, p, q\}$: Then $\{p, q\}$ must be a module of $G[V - \{a, b, w\}]$ which is contrary to the above mentioned claim.

(ii) $\{a, u\}$: Then $\{q, u\}$ is a module of $G[V - \{a, b, w\}]$. This implies that $e_{pu} = 1$ because $e_{pq} = 1$. But $e_{pu} = e_{bu}$ so $e_{bu} = 1$, while $e_{ba} = 0$ so $\{a, u\}$ cannot be a module in $G[V - \{w\}]$.

(iii) $\{q, u\}$: then it is also a module of $G[V - \{a, b, w\}]$.

(iv) $\{q, u, b\}, \{q, u, a, b\}$: Not possible because $\{p, q, u\}$ is a module of $G[V - \{a, b, w\}]$ with cardinality different form 2 and $|V| - 4$.

(v) $\{u, v\}$: It is also the module of $G[V - \{a, b, w\}]$.

(vi) $\{u, v, a\}$: Not possible since $\{u, v, q\}$ (improper cardinality) is module of $G[V - \{a, b, w\}]$.

(vii) $\{u, v, a, b\}$: Not possible since $\{u, v, q, p\}$ (improper cardinality) is module of $G[V - \{a, b, w\}]$.
(viii) $V-\{a, b, u, w\}$: Not possible because $V-\{a, b, p, u, w\}$ and $V-\{a, b, q, u, w\}$ are both modules (nontrivial because $|V| > 6$) of $G[V - \{a, b, w\}]$.
(ix) $V - \{u, w\}$: Then $V - \{a, b, u, w\}$ is a module of $G[V - \{a, b, w\}]$.
(x) $V - \{a, b, w\}, V - \{a, b, p, w\}, V - \{b, w, q\}$: Then $V - \{a, b, p, q, w\}$ should be a module of $G[V - \{a, b, w\}]$ which is impossible due to its size.
(xi) $V-\{b, w, u\}$: Again this is impossible because it implies that $V-\{a, b, p, u, w\}$ is a module of $G[V - \{a, b, w\}]$ and its size is improper.
(xii) $V - \{w, p\}$: Then $V - \{a, b, p, w\}$ will be the module of $G[V - \{a, b, w\}]$.

The summary of feasible cases is as follows. $G[V-\{a, b, w\}]$ must have module M' whenever $G[V - \{w\}]$ has module M:
(i) $M' = \{q, u\}$ for $M = \{q, u\}$,
(ii) $M' = \{p, u\}$ for $M = \{p, u\}$,
(iii) $M' = \{u, v\}$ for $M = \{u, v\}$,
(iv) $M' = V - \{a, b, u, w\}$ for $M = V - \{u, w\}$
(v) $M' = V - \{a, b, w, p\}$ for $M = V - \{w, p\}$
(vi) $M' = V - \{a, b, w, q\}$ for $M = V - \{w, q\}$.

If $G[V - \{w\}]$ had more than one (non-trivial) module, then $G[V - \{a, b, w\}]$ will have to have more than one (non-trivial) module which is not possible. Besides each module has cardinality 2 or $|V| - 2$ so $G[V - \{w\}]$ is marginally decomposable for each w.

Table 2 shows all the possible modules, M, of $G[V-\{w\}]$ and implied module, M', of $G[V - \{a, b, w\}]$ in the corresponding case.

Table 2. Case of a and b both in $G[V - \{a, b\}]$-Equi

w	M	M′	Remark
a	$\{b, p\}$		M' can't exist as $w \notin V - \{a, b\}$
b	$\{a, q\}$		M' can't exist as $w \notin V - \{a, b\}$
p	$\{a, u\}$ $V - \{a, p\}$	$\{q, u\}$ $V - \{a, b, q, p\}$	u is some vertex of $V - \{a, b, p, q\}$
q	$\{b, u\}$ $V - \{b, q\}$	$\{p, u\}$ $V - \{a, b, q, p\}$	u is some vertex of $V - \{a, b, p, q\}$
w	 $V - \{x, w\}$ $\{v, x\}$	 $V - \{a, b, x, w\}$ $\{v, x\}$	for any w in $V - \{a, b, p, q\}$ x is some vertex of $V - \{a, b, w\}$ x is some vertices of $V - \{a, b, w\}$ and v is some vertex of $V - \{a, b, p, q, x, w\}$

The main result of this section follows.

Theorem 2. *Let $G = (V, E)$ is X-ci graph with $|X| \geq 3$. Then G is X-si.*

Proof. Let Y be a variables that denotes a subset of V such that $G[Y]$ is X-ci. We will prove that $G[Y]$ is X-si for all Y by induction on the size of $Y - X$.

As a base case we will consider Y such that $|Y - X| \leq 3$. From Corollary 1 we know that $|Y - X|$ cannot be odd. The case of $|Y - X| = 0$ is vacuously

true. If $|Y - X| = 2$ then let $Y = X \cup \{a, b\}$. From lemma 1(a), $G[X \cup \{a\}]$ and $G[X \cup \{b\}]$ are marginally decomposable so $G[Y]$ is X-si.

Now we consider the induction step. From hypothesis every X-ci subgraph with upto k more vertices than $|X|$, is X-si. Due to the base case analysis k is no smaller than 3. Now consider Y s.t. $|Y - X| = k + 1$. Since $X \geq 3$, $|Y| > 6$. Lemmas 3, 5, 6, and Proposition 2 lead to the conclusion that $G[Y]$ is X-si. So from induction principle $G[Y]$ is X-si for all Y.

4 A Commutative Elimination Sequence

Proposition 3. *G is X-ci graph. Then (a, b) is a locked pair of G iff $a, b \in V - X$ and the (unique) module of $G[V - \{b\}]$ is $V - \{a, b\}$ or $\{a, q\}$ for some $q \in V - \{a, b\}$.*

Proof. (if) This is the consequence of Proposition 1(c). (only if) From tables 1, 2 we know that only possible module in $G[V - \{b\}]$ can be either $V - \{a, b\}$ or $\{a, q\}$.

Lemma 7. *G is X-ci. (a, b) and (c, d) are locked pairs in G with no common vertex. Then (a, b) is also a locked pair in $G[V - \{c, d\}]$*

Proof. At least one of a and b is in $G[V - \{a, b\}]$-Equi. Without loss of generality assume that $\{a, q\}$ is the module of $G[V - \{b\}]$. If $q \notin \{c, d\}$, then $\{a, q\}$ is the module in $G[V - \{b, c, d\}]$. From Proposition 3, (a, b) is a locked pair in $G[V - \{c, d\}]$.

Now consider the case $q \in \{c, d\}$. Without loss of generality $q = c$. Consider the module of $G[V - \{d\}]$. From Proposition 3, it is either $V - \{c, d\}$ or $\{c, e\}$ for some $e \in V - \{c, d\}$. In the former case $V - \{c, d, a, b\}$ is a module of $G[V - \{c, d, b\}]$ (because $\{a, c\}$ is a module of $G[V - \{b\}]$) so (a, b) is a locked pair in $G[V - \{c, d\}]$.

The latter case is as follows. $\{a, c\}$ is the module in $G[V - \{b\}]$, $\{c, e\}$ is the module in $G[V - \{d\}]$. If $e = a$ then $\{a, c\}$ will be a module of G which is not possible. If $e = b$ then $\{a, b, c\}$ will be a module of $G[V - \{d\}]$ which is not possible since $G[V - \{d\}]$ is marginally decomposable. Thus we find that $e \in V - \{a, b, c, d\}$. Then $\{a, e\}$ is the module of $G[V - \{c, d, b\}]$. Once again (a, b) is a locked pair in $G[V - \{c, d\}]$.

From Corollary 1 we know that in an X-ci graph $V - X$ has even number of vertices, say $2k$. If these vertices can be partitioned into k 2-tuples such that each tuple is a locked pair of the graph then such a collection is called a *commutative elimination sequence.*

Lemma 8. *Let G be X-ci and (a, b) be a locked pair in it. If $G[V - \{a, b\}]$ has a commutative elimination sequence, then so does G.*

Proof. Suppose E' is a commutative elimination sequence of $G[V - \{a, b\}]$. We will first show that at most one locked pair in E' may not remain a locked pair of G.

Consider any locked pair $(c, d) \in E'$ in which at least one vertex w is such that $\{w, a\}$ is not a module in $G[V - \{b\}]$ and $\{w, b\}$ is not a module in $G[V - \{a\}]$. Let v be the other vertex of $\{c, d\}$. The module M' of $G[V - \{a, b, w\}]$ must be either $V - \{a, b, w, v\}$ or $\{v, x\}$ for some x because $G[V - \{a, b, w\}]$ is marginally decomposable. From tables 1, 2 we find that the module M of $G[V - \{w\}]$ should be $M' \cup \{a, b\}$ if $M' = V - \{a, b, w, v\}$ and $M = M'$ if $M' = \{v, x\}$. Thus M is either $V - \{w, v\}$ or $\{v, x\}$.

From Proposition 3 we deduce that (w, v) (which is same as (c, d)) remains a locked pair in G. If all pairs on E' are found to remain locked pairs in G, then $E = E' \cup \{(a, b)\}$ is a commutative elimination sequence of G.

In case all locked pairs of $G[V - \{a, b\}]$ are not locked pairs of G then the exception must be only one pairs (p, q) where $\{a, q\}$ is the module of $G[V - \{b\}]$ and $\{b, p\}$ is the module of $G[V - \{a\}]$. From Table 2, if $\{q, u\}$ is the module of $G[V - \{a, b, p\}]$, then $\{a, u\}$ is the module of $G[V - \{p\}]$. Thus (a, p) is a locked pair of G. Similar argument shows that (b, q) is a locked pair of G. In this case $E = E' - \{(p, q)\} \cup \{(a, p), (b, q)\}$ is a commutative elimination sequence.

Vacuously, the subgraph $G[X]$ of X-ci G has a commutative elimination sequence. By induction and using lemma 8 we have a trivial conclusion that every X-ci graph has a commutative elimination sequence. Now from lemma 7 we have the following theorem.

Theorem 3. *If G is X-ci, then vertices of $V - X$ can be partitioned into pairs $(a_1, b_1), \ldots, (a_k, b_k)$ such that for any permutation $j_1, \ldots j_k$ of $1, \ldots, k$, $G[V - \{a_{j_1}, b_{j_1}, \ldots, a_{j_s}, b_{j_s}\}]$ is also X-ci.*

References

1. Schmerl, J.H., Trotter, W.T.: Critically indecomposable partially ordered sets, graphs, tournaments and other binary relational structures. Discrete Mathematics **113** (1995) 191–205
2. Cournier, A., Ille, P.: Minimal indecomposable graphs. Discrete Mathematics **183** (1998) 61–80
3. Ille, P.: Indecomposable graphs. Discrete Mathematics **173** (1997) 71–78
4. Ehrenfeucht, A., Rozenberg, G.: Primitivity is hereditry for 2-structures. Theoritical Computer Science **3** (1990) 343–358
5. Spinrad, J.: p_4-trees and substitution decomposition. Discrete Applied Math. **39** (1992) 263–291
6. Cournier, A., Michel, H.: An efficient algorithm to recognize prime undirected graphs. Lecture Notes in Computer Science **657** (1992) 212–224

A Tight Analysis of the Maximal Matching Heuristic

Jean Cardinal[1], Martine Labbé[1], Stefan Langerman[1,⋆], Eythan Levy[1,⋆⋆], and Hadrien Mélot[2]

[1] Université Libre de Bruxelles, Brussels, Belgium
{jcardin,mlabbe,slanger,elevy}@ulb.ac.be
[2] Université de Mons-Hainaut, Belgium
hadrien.melot@umh.ac.be

Abstract. We study the worst-case performance of the maximal matching heuristic applied to the Minimum Vertex Cover and Minimum Maximal Matching problems, through a careful analysis of tight examples. We show that the tight worst-case approximation ratio is asymptotic to $\min\{2, 1/(1-\sqrt{1-\epsilon})\}$ for graphs with an average degree at least ϵn and to $\min\{2, 1/\epsilon\}$ for graphs with a minimum degree at least ϵn.

1 Introduction

The maximal matching heuristic is a textbook algorithm that provides a 2-approximation for the Minimum Vertex Cover and Minimum Maximal Matching problems, two classical NP-hard problems [8]. It is perhaps one of the simplest and best-known approximation algorithms. It consists in finding a collection of disjoint edges (a matching) that is maximal (with respect to edge inclusion) by iteratively removing adjacent vertices until no more edges are left in the graph. Tightness of the 2-approximation is witnessed by a number of examples, for instance by the family of complete bipartite graphs in the case of Minimum Vertex Cover. This paper addresses the question of expressing the approximation ratio in a finer way, as a function of well-chosen graph parameters. We show that density parameters are good candidates for this purpose. Actually, the approximation ratio of the maximal matching heuristic is strictly less than 2 for graphs with a sufficiently high number of edges or sufficiently high minimum degree. We characterize precisely the asymptotic approximation ratio as a function of these parameters, together with tight examples. This is, to our knowledge, the tightest analysis ever done of this algorithm. This study shows that even simple heuristics might deserve nontrivial analyses. It was initiated using GraPHedron, a newly developed software for the investigation of relations between graph invariants (see [4] and [16]).

In the Minimum Vertex Cover problem, one is asked to find a minimum cardinality set of vertices that contains at least one endpoint of each edge of

⋆ Chercheur qualifié du F.N.R.S.
⋆⋆ Corresponding author.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 701–709, 2005.

the graph. One can easily see that the set of endpoints of a maximal matching indeed contains at least one endpoint of each edge, and that the optimal solution – since it must contain a least one vertex of each edge of the matching – has size not less than half the size of this set, hence the 2-approximation.

The minimum vertex cover problem is thus 2-approximable using the maximal matching heuristic, but no polynomial time algorithm with constant approximation ratio better than 2 is known. The problem is further known to be APX-complete [18] and not approximable within a factor of 7/6 [11]. Monien and Speckenmeyer [17] and Bar-Yehuda and Even [1] provide algorithms that achieves a ratio of $(2 - (\ln \ln n)/\ln n)$, where n is the number of vertices in the graph. Karakostas [14] later reduced the approximation ratio to $2-\Theta(1/\sqrt{\log n})$. For graphs with maximum degree Δ, Halperin [9] provides an aproximation algorithm with a ratio of $2-(1-o(1))2\ln\ln\Delta/\ln\Delta$. The problem has further been studied under the hypothesis that the input graph is dense. We say that a graph G is *weakly* ϵ*-dense* if its *average* degree is at least ϵn, i.e. if $m \geq \epsilon n^2/2$, with m being the number of edges in the graph, and *strongly* ϵ*-dense* if its *minimum* degre is at least ϵn. It has been shown [5] that the Minimum Vertex Cover problem restricted to strongly ϵ-dense graphs is APX-complete. Eremeev [7] shows that it is NP-hard to approximate the minimum vertex cover within a ratio less than $(7+\epsilon)/(6+2\epsilon)$ in strongly ϵ-dense graphs. Nagamochi and Ibaraki ([12]) provide an approximation algorithm with a ratio of $2 - 8m/(13n^2 + 8m)$, where m is the number of edges in the graph. This algorithm can also be seen as an approximation algorithm that achieves an approximation ratio that is asymptotic to $2-4\epsilon/(13+4\epsilon)$ for weakly ϵ-dense graphs. Karpinski and Zelikovsky [15] propose an algorithm that achieves a better ratio of $2/(2-\sqrt{1-\epsilon})$ for weakly ϵ-dense graphs, and a ratio of $2/(1+\epsilon)$ for strongly ϵ-dense graphs. Finally, Imamura and Iwama [13] recently proposed a randomized approximation algorithm which, with high probability, yields an approximation factor of $2/(1+\gamma(G))$, where $\gamma(G)$ is a function of the maximum and the average degree, and runs in polynomial time if Δ, the maximum degree of the graph, is $\Omega(n \log\log n/\log n)$.

In the Minimum Maximal Matching problem, one is asked to find a maximal matching of minimum cardinality, i.e. a minimum-cardinality set of disjoint edges that cannot be augmented. It is fairly easy to see that any maximal matching has a size that is at most twice the size of the minimum maximal matching. Much less is known about the minimum maximal matching problem than about the minimum vertex cover problem. Chlebìk and Chlebìková [3] do nevertheless show that it is NP-hard to approximate the problem within a constant factor better than 7/6.

Our Results. We study the worst-case approximation ratio of the maximal matching heuristic for the Minimum Vertex Cover and Minimum Maximal Matching problems in weakly and strongly ϵ-dense graphs. For both problems in weakly ϵ-dense graphs, we characterize the exact worst-case approximation ratio as a function of ϵ and obtain a function that is asymptotic to 2 when $\epsilon \leq 3/4$ and to $1/(1-\sqrt{1-\epsilon})$ otherwise. In the case of strongly ϵ-dense graphs, again we characterize the exact worst-case approximation ratio as a function of ϵ and

obtain a function that is asymptotic to 2 when $\epsilon \leq 1/2$ and to $1/\epsilon$ otherwise. It is interesting to compare the approximation ratios we obtain for Minimum Vertex Cover with the ones obtained by Zelikovsky and Karpinski: we note that in both the weakly and the strongly ϵ-dense cases the ratios differ only by one unit in the numerator and one in the denominator. The tight bounds we obtain for the Minimum Vertex Cover problem are greater than those that were obtained using more sophisticated approximation algorithms [12, 15]. We nevertheless believe that a tight worst-case study of the classical heuristic is interesting as a point of comparison to the ratios obtained by other algorithms for the same problem, or by the same heuristic applied to other problems. On the other hand, the approximation ratios obtained for Minimum Maximal Matching are, to the best of our knowledge, the best ones known for this problem since it does not seem to have been studied under density constraints yet. Finally, the results obtained for this problem are also valid for a variant problem, namely the Minimum Edge Dominating Set problem since, as noted by by Yannakakis and Gavril [19], both problems always admit optimal solutions of the same size, and an optimal solution to one can always be transformed into an optimal solution to the other in polynomial time.

Section 2 is devoted to graph-theoretic preliminaries. In section 3 we study the worst-case aproximation ratio for the Minimum Vertex Cover problem in weakly and strongly ϵ-dense graphs. The same kind of analysis is performed for Minimum Maximal Matching in section 4. Full proofs are omitted through lack of space and can be found in [2].

2 Preliminaries

In the sequel we shall use the classical definition of a simple, loopless, undirected graph $G = (V, E)$, with vertex set V and edge set E. We denote by $\mathcal{G}_{n,m}$ the set of all non isomorphic graphs having n vertices and m edges. We use $n(G)$ to denote $|V|$, $m(G)$ to denote $|E|$ and $\delta(G)$ for the minimum degree of G. We also use the classical notions of *complete graph*, *empty graph*, *independent set*, *clique*, *complete bipartite graph* $K_{a,b}$, *matching*, *perfect matching* and *augmenting path*. Readers that are not familiar with these are referred to standard graph theory texts such as Diestel [6]. The *join* of two graphs G_1 and G_2 with vertex sets respectively V_1 and V_2 is the graph having $V_1 \cup V_2$ as vertex set and containing all edges of G_1, G_2, and all edges between vertices in V_1 and V_2. We denote by $\tau(G)$ the size of a minimum cardinality vertex cover of G, by $\nu(G)$ the size of a maximum cardinality matching of G, and by $\mu(G)$ the size of a minimum maximal matching of G.

We shall make extensive use of the following family of graphs, that arise as extremal graphs for several graph invariants (see [10] and [4]). A *complete split graph* $\Psi_{n,\alpha}$ with $1 \leq \alpha \leq n-1$, is a graph that can be decomposed in an independent set of size α and a clique of size $n - \alpha$, with each vertex of the independent set being adjacent to each vertex in the clique.

Our proofs make use of the following basic results on the values of invariants of complete split graphs.

Lemma 2.1. $m(\Psi_{n,\alpha}) = \binom{n}{2} - \binom{\alpha}{2} = (n-\alpha)(n+\alpha-1)/2$.

Lemma 2.2. $\nu(\Psi_{n,\alpha}) = \begin{cases} \lfloor \frac{n}{2} \rfloor & \text{if } \alpha \leq n/2, \\ n-\alpha & \text{otherwise.} \end{cases}$

Lemma 2.3. $\tau(\Psi_{n,\alpha}) = n - \alpha$.

Note that $\tau(\Psi_{n,\alpha}) = n - \alpha$ is obtained only by choosing all vertices in the clique as a vertex cover.

Lemma 2.4. $\mu(\Psi_{n,\alpha}) = \lceil \frac{n-\alpha}{2} \rceil$.

The following two simple properties shall also be useful.

Lemma 2.5. *For any graph G, we have $\tau(G) \geq \delta(G)$.*

Lemma 2.6. *For any graph G, we have $\mu(G) \geq \left\lceil \frac{\delta(G)}{2} \right\rceil$.*

3 Minimum Vertex Cover

We analyze the worst-case behavior of the maximal matching heuristic when applied to the Minimum Vertex Cover problem. We first consider weakly ϵ-dense graphs, which amounts to express the approximation ratio as a function of the number of edges.

3.1 Approximation Ratio vs Number of Edges

Lemma 3.1. *Let n and m be positive integers such that $m(\Psi_{n,\alpha+1}) < m \leq m(\Psi_{n,\alpha})$ for some α. The minimum value of $\tau(G)$ attained by a graph G in $\mathcal{G}_{n,m}$ is $\tau(\Psi_{n,\alpha}) = n - \alpha$.*

Sketch of the proof. The proof is in two steps: we first show by contradiction that a graph G in $\mathcal{G}_{n,m}$ cannot have $\tau(G) < \tau(\Psi_{n,\alpha})$, and second, by construction, that there exists a graph G in $\mathcal{G}_{n,m}$ having $\tau(G) = \tau(\Psi_{n,\alpha})$. □

Lemma 3.2. *Let n and m be positive integers such that $m(\Psi_{n,\alpha+1}) < m \leq m(\Psi_{n,\alpha})$ for some α. There exists a graph G in $\mathcal{G}_{n,m}$ such that $\tau(G) = \tau(\Psi_{n,\alpha})$ and $\nu(G) = \nu(\Psi_{n,\alpha})$.*

Sketch of the proof. A graph satisfying the required conditions is given by the removal of $m(\Psi_{n,\alpha}) - m$ edges from the edges joining the clique and the independent set of $\Psi_{n,\alpha}$. This graph can be shown to satisfy the required conditions by the use of classical tools, among wich Hall's condition on perfect matchings (see [6]). □

Using Lemmata 3.1 and 3.2 to maximize the numerator and minimize the denominator of the ratio, we obtain Theorem 3.1:

Theorem 3.1. *Let $\beta(G)$ be the worst-case approximation ratio for graph G. Let $\beta(m,n)$ be the worst approximation ratio attained by a graph in $\mathcal{G}_{n,m}$. We have:*

$$\beta(m,n) = \beta(\Psi_{n,\alpha^*(m,n)}) = \begin{cases} 2 & \text{if } \alpha^*(m,n) > n/2, \\ \frac{2\lfloor \frac{n}{2} \rfloor}{n-\alpha^*(m,n)} & \text{otherwise,} \end{cases}$$

where $\alpha^*(m,n) = \left\lfloor 1/2 + \sqrt{n(n-1) + 1/4 - 2m} \right\rfloor$ *is the integer value α such that* $m(\Psi_{n,\alpha+1}) < m \leq m(\Psi_{n,\alpha})$.

The above theorem gives a tight upper bound on the approximation ratio of the maximal matching heuristic to the minimum vertex cover problem in graphs of n vertices and m edges, in the form of a discrete step function of m. The function equals 2 when $\alpha > n/2$ and begins to decrease afterwards.

Corollary 3.1. *Let $\tilde{\beta}(\epsilon,n)$ be the worst approximation ratio attained by a graph with n vertices and an average degree at least ϵn. We have:*

$$\lim_{n\to\infty} \tilde{\beta}(\epsilon,n) = \begin{cases} 2 & \text{if } \epsilon \leq 3/4, \\ \frac{1}{1-\sqrt{1-\epsilon}} & \text{otherwise .} \end{cases}$$

Sketch of the proof. The corollary follows from expressing $\tilde{\beta}(\epsilon,n)$ as $\beta(\lceil \epsilon n^2/2 \rceil, n)$ and studying the asymptotics of this expression. □

This asymptotic result is to be compared with the results of [12] and [15] quoted in the introduction (see figure 1).

3.2 Approximation Ratio vs Minimum Degree

Let $A_{n,\alpha}$ be the set of all graphs of minimum degree $n-\alpha$ that can be expressed as the join of an independent set of order α and a graph of order $n-\alpha$. Note that $A_{n,\alpha}$ contains $\Psi_{n,\alpha}$.

Lemma 3.3. *For all $G \in A_{n,\alpha}$ we have* $\nu(G) = \begin{cases} n-\alpha & \text{if } \alpha \geq n/2, \\ \lfloor \frac{n}{2} \rfloor & \text{otherwise.} \end{cases}$

Sketch of the proof. When $\alpha \geq n/2$, the result is straightforward. When $\alpha < n/2$, through a careful analysis of the configurations and degrees of unmatched vertices, we show that any non-perfect matching can be augmented. □

Lemma 3.4. *For all $G \in A_{n,\alpha}$ we have $\tau(G) = n-\alpha$. Furthermore, among all graphs with n vertices and minimum degree $n-\alpha$, this value of τ is minimal, and is attained only by graphs in $A_{n,\alpha}$.*

Sketch of the proof. The proof follows from a direct application of Lemma 2.5 and simple graph-theoretic arguments. □

Theorem 3.2 follows from Lemmata 3.3 and 3.4:

Theorem 3.2. *Let $\gamma(\delta, n)$ be the worst approximation ratio attained by a graph with n vertices and minimum degree δ. We have:*

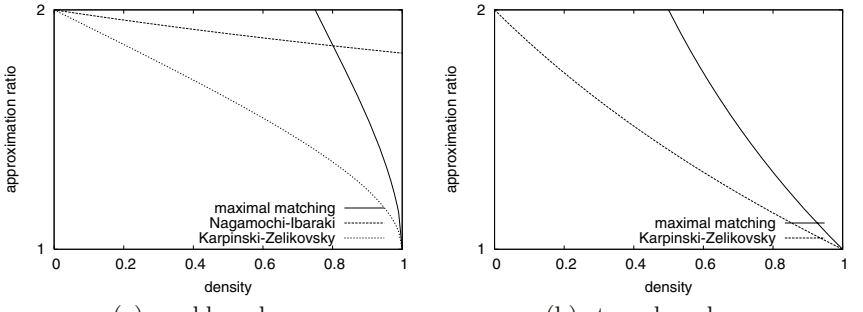
$$\gamma(\delta, n) = \begin{cases} 2 & \text{if } \delta \leq \frac{n}{2}, \\ \frac{2}{\delta}\lfloor \frac{n}{2} \rfloor & \text{otherwise.} \end{cases}$$

Furthermore, the only graphs that maximize the approximation ratio among all graphs with n vertices and a minimum degree of $n-\alpha$ when $\delta > n/2$ are in $A_{n,\alpha}$.

Corollary 3.2. *Let $\tilde{\gamma}(\epsilon, n)$ be the worst approximation ratio attained by a graph with n vertices and minimum degree at least ϵn. We have:*

$$\lim_{n\to\infty} \tilde{\gamma}(\epsilon, n) = \begin{cases} 2 & \text{if } \epsilon \leq 1/2, \\ \frac{1}{\epsilon} & \text{otherwise .} \end{cases}$$

This asymptotic result is again to be compared with the result of [15] quoted in the introduction (see figure 1).



(a) weakly ϵ-dense (b) strongly ϵ-dense

Fig. 1. A comparison of the approximation ratios for Minimum Vertex Cover.

4 Minimum Maximal Matching

In this section we show that the analysis proposed for Minimum Vertex Cover can be performed for Minimum Maximal Matching as well.

4.1 Approximation Ratio vs Number of Edges

The proof of the following lemma is straightforward and omitted.

Lemma 4.1. *For any fixed k, the only graph G with n vertices that maximizes the number of edges among all graphs having $\mu(G) = k$ is $\Psi_{n,n-2k}$.*

Lemma 4.2. *Let n and m be positive integers such that $m(\Psi_{n,\alpha+1}) < m \leq m(\Psi_{n,\alpha})$ for some α. The minimum value for $\mu(G)$ attained by a graph G in $\mathcal{G}_{n,m}$ is $\mu(\Psi_{n,\alpha}) = \lceil \frac{n-\alpha}{2} \rceil$.*

Sketch of the proof. Using arguments similar to those used in the proofs of Lemmata 3.1 and 3.2, we first show, using Lemma 4.1, that a graph in $\mathcal{G}_{n,m}$ cannot have a maximal matching of size less than $\mu(\Psi_{n,\alpha})$, and then that the value $\mu(\Psi_{n,\alpha})$ is indeed attained by some graph in $\mathcal{G}_{n,m}$. □

Lemma 4.3. *Let n and m be positive integers such that $m(\Psi_{n,\alpha+1}) < m \leq m(\Psi_{n,\alpha})$ for some α. There exists a graph G in $\mathcal{G}_{n,m}$ such that $\mu(G) = \mu(\Psi_{n,\alpha})$ and $\nu(G) = \nu(\Psi_{n,\alpha})$.*

Sketch of the proof. The graphs described in the proof of Lemma 3.2 can easily be shown to satisfy $\mu(G) = \mu(\Psi_{n,\alpha})$ and $\nu(G) = \nu(\Psi_{n,\alpha})$. □

Using Lemmata 4.2 and 4.3 to maximize the numerator and minimize the denominator of the ratio, we obtain Theorem 4.1:

Theorem 4.1. *Let $\rho(G)$ be the worst approximation ratio for graph G. Let $\rho(m,n)$ be the worst approximation ratio attained by a graph in $\mathcal{G}_{n,m}$. For each n, m we have:*

$$\rho(m,n) = \rho(\Psi_{n,\alpha^*(m,n)}) = \begin{cases} 2 & \text{if } \alpha^*(m,n) > n/2+1, \\ \frac{\lfloor \frac{n}{2} \rfloor}{\left\lceil \frac{n-\alpha^*(m,n)}{2} \right\rceil} & \text{otherwise.} \end{cases}$$

Corollary 4.1. *Let $\tilde{\rho}(\epsilon, n)$ be the worst approximation ratio attained by a graph with n vertices and an average degree at least ϵn. We have:*

$$\lim_{n\to\infty} \tilde{\rho}(\epsilon,n) = \begin{cases} 2 & \text{if } \epsilon \leq 3/4, \\ \frac{1}{1-\sqrt{1-\epsilon}} & \text{otherwise.} \end{cases}$$

4.2 Approximation Ratio vs Minimum Degree

Let $B_{n,\delta}$ be the set of graphs of order n having minimum degree δ and a maximal matching of size $\lceil \delta/2 \rceil$. Note that $B_{n,\delta} = A_{n,n-\delta}$ when δ is even.

Lemma 4.4. *Each graph in $B_{n,\delta}$ with $\lceil \delta/2 \rceil > n/4$ has a perfect matching.*

Sketch of the proof. As in the proof of Lemma 3.3, we show that any non-perfect matching in our graph can be augmented, using a careful analysis of the configuration and degrees of the vertices. □

Theorem 4.2 follows directly from Lemmata 2.6 and 4.4:

Theorem 4.2. *Let $\sigma(\delta, n)$ be the worst approximation ratio attained by a graph with n vertices and minimum degree δ. We have:*

$$\sigma(\delta,n) = \begin{cases} 2 & \text{if } \lceil \delta/2 \rceil \leq n/4 \\ \frac{\lfloor n/2 \rfloor}{\lceil \delta/2 \rceil} & \text{otherwise.} \end{cases}$$

Furthermore, when $\lceil \delta/2 \rceil > n/4$, $B_{n,\delta}$ is the exact set of graphs that maximize the ratio among all graphs with n vertices and minimum degree δ.

Corollary 4.2. *Let $\tilde{\sigma}(\epsilon, n)$ be the worst approximation ratio attained by a graph with n vertices and minimum degree at least ϵn. We have:*

$$\lim_{n\to\infty} \tilde{\sigma}(\epsilon, n) = \begin{cases} 2 & \text{if } \epsilon \leq 1/2, \\ \frac{1}{\epsilon} & \text{otherwise.} \end{cases}$$

References

1. R. Bar-Yehuda and S.Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Ann. Discrete Math.*, 25:27–45, 1985.
2. J. Cardinal, M. Labbé, S. Langerman, E. Levy, and H. Mélot. A tight analysis of the maximal matching heuristic. Technical Report 545, Université Libre de Bruxelles, 2005. Available at http://www.ulb.ac.be/di/publications.
3. M. Chlebìk and J. Chlebìková. Approximation hardness of minimum edge dominating set and minimum maximal matching. In *Proceedings of 14th International Symposium on Algorithms and Computation (ISAAC)*, Lecture Notes in Computer Science. Springer–Verlag, 2003.
4. J. Christophe, S. Dewez, J. Doignon, S. Elloumi, G. Fasbender, P. Grégoire, D. Huygens, M. Labbé, H. Mélot, and H. Yaman. Linear inequalities among graph invariants: using GraPHedron to uncover optimal relationships. Submitted. Available at http://www.optimization-online.org/DB_HTML/2004/09/964.html, 2004.
5. A. Clementi and L. Trevisan. Improved non-approximability results for vertex cover problems with density constraints. *Theoretical Computer Science*, 225(1–2):113–128, 1999.
6. R. Diestel. *Graph Theory, Second Edition*. Springer-Verlag, 2000.
7. A.V. Eremeev. On some approximation algorithms for dense vertex cover problem. In *Proceedings of SOR*. Springer–Verlag, 1999.
8. M. R. Garey and D. S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. Freeman and Company, 1979.
9. E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *Siam Journal on Computing*, 31:1608–1623, 2002.
10. P. Hansen and H. Mélot. Variable neighborhood search for extremal graphs 9. Bounding the irregularity of a graph. To appear in S. Fajtlowicz *et al.* (Eds.), Graphs and Discovery, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, (forthcoming), Providence, American Mathematical Society.
11. J. Hastad. Some optimal inapproximability results. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 1–10, 1997.
12. T. Ibaraki and H. Nagamochi. An approximation of the minimum vertex cover in a graph. *Japan J. Indust. Appl. Math.*, 16:369–375, 1999.
13. T. Imamura and K. Iwama. Approximating vertex cover on dense graphs. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005.
14. G. Karakostas. A better approximation ratio for the vertex cover problem. *ECCC Report TR04-084*, 2004.
15. M. Karpinski and A. Zelikovsky. Approximating dense cases of covering problems. In P. Pardalos and D. Du, editors, *Proceedings of the DIMACS Workshop on Network Design: Connectivity and Facilites Location*, volume 40 of *DIMACS series in Disc. Math. and Theor. Comp. Sci.*, pages 169–178, 1997.
16. H. Mélot. Facets defining inequalities among graph invariants: the system GraPHedron. In preparation.

17. B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Inf.*, 22:115–123, 1985.
18. C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. System. Sci.*, 43(3):425–440, 1991.
19. M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM J. Appl. Math.*, 38(3):364–372, 1980.

New Streaming Algorithms for Counting Triangles in Graphs

Hossein Jowhari and Mohammad Ghodsi

Computer Engineering Department
Sharif University of Technology, Tehran, Iran
jowhari@ce.sharif.edu, ghodsi@sharif.edu

Abstract. We present three streaming algorithms that $(\epsilon, \delta)-$ approximate[1] the number of triangles in graphs. Similar to the previous algorithms [3], the space usage of presented algorithms are inversely proportional to the number of triangles while, for some families of graphs, the space usage is improved. We also prove a lower bound, based on the number of triangles, which indicates that our first algorithm behaves almost optimally on graphs with constant degrees.

1 Introduction

In this paper, we present streaming algorithms for counting triangles in massive graphs. In other words, let $G = (V, E)$ be an undirected graph with n vertices and m edges and let t be the number of triangles in G. we are interested in algorithms with sublinear space usage for $(\epsilon, \delta)-$approximating t while G is presented to the algorithm as a stream of edges. By sublinear space usage, we mean algorithms that use $o(m)$ bit space, and by stream of edges, we mean a sequence of edges that is an arbitrary permutation of E. In addition to the space usage, we restrict the algorithms to have only $O(1)$ passes over the stream and $o(m)$ per-edge processing time.

Bar-Yossef *et al* in [3] showed that every algorithm that decides the existence of a triangle, with probability at least 99/100, needs at least $\Omega(n^2)$ bit space. This fact results in a lower bound of $\Omega(n^2)$ for $(\epsilon, \delta)-$approximating t in general graphs, but Bar-Yossef *et al* showed that for graphs with considerably large number of triangles, it is possible to gain sublinear space. In other words, let T_i be the number of vertex triples that induce a subgraph with i edges in G. Based on this definition, $T_3 = t$. For $T_3 > 0$, they obtained a streaming algorithm with $O(1/\epsilon^3 . \log 1/\delta . ((T_1 + T_2 + T_3)/T_3)^3 . \log n)$ space. Since $(T_1 + T_2 + T_3) = \Theta(mn)$, having an appropriate lower bound for T_3, one can use $o(m)$ space on graphs with $m^{2/3}n = o(T_3)$.

Our Contribution. We present three streaming algorithms for (ϵ, δ)-approximating T_3. Let d be the maximum degree and let C_i be the number of cycles

[1] Let $\epsilon, \delta > 0$ and let T_3 be the number of triangles. With probability at least $1 - \delta$, the algorithm outputs T_3' such that $(1 - \epsilon)T_3 \leq T_3' \leq (1 + \epsilon)T_3$.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 710–716, 2005.

of length i in the input graph. The first algorithm uses $O(1/\epsilon^2.\log(1/\delta).(md^2)/T_3.\log n)$ space and per-edge processing time and makes one pass over the stream (Theorem 1). The second algorithm uses $O(1/\epsilon^2.\log(1/\delta).(m^3 + mC_4 + C_6 + {T_3}^2)/{T_3}^2.\log n)$ space and per-edge processing time with one pass (Theorem 2) and the third algorithm uses $O(n + 1/\epsilon^2.\log(1/\delta).(T_2 + T_3)/T_3.\log n)$ while making three passes over the stream (Theorem 3). The first and second algorithm use the method of Alon *et al* in [2] and the third algorithm utilizes sampling to reduce the space usage. We also prove a lower bound of $\Omega(n/T_3)$ that indicates that our first algorithm behaves almost optimally when d is a constant (Theorem 4).

Related Work. After the seminal paper by Alon *et al* [2], Henzinger *et al* [7] formalized the streaming model and proved lower bounds for some graph problems. According to our knowledge, there are few attempts for solving graph problems in the streaming model. Recently, authors in [5, 6] have presented streaming algorithm for some graph problems such as maximum weighted matching and shortest path. In fact, most of the results for graph problems are impossibility results [4–7].

2 Algorithms

First, we define some notations. For undirected graph $G = (V, E)$, let n, m and d be the number of vertices, edges and maximum degree respectively. Let Υ be the set of all vertex triples of G. We partition Υ to four parts Υ_i, $i = 0, \ldots, 3$, where Υ_i is the set of triples that induces a subgraph in G with i total edges and let $T_i = |\Upsilon_i|$. For $j = 4, \ldots, n$, let C_j be the number of cycles of length j in G .

2.1 One-Pass Algorithms

Here we present two algorithm that uses the method of Alon et al [2]. We define random variable X such that $E(X) = T_3$. By taking the average of an appropriate number of independent instances of X, we can reduce the variance so that by using Chebyshev's Inequality, we can obtain an approximation for T_3 with relative error at most ϵ. Hence, the space usage of the algorithm depend on $Var(X)$ and the space requirements for computing the random variable X.

First Estimator. The random variable X is computed as follows. Choose an edge (u, v), randomly and uniformly from the edges in the stream. Count the number of common neighbors of vertices u and v in the rest of the stream. Let c be the value of this counter at the end of the stream. We define $X = mc$.

Now, we compute the expectation and variance of X. Suppose we have an ordering on the triangles. For i-th triangle, we define indicator random variable Z_i as follows.

$$Z_i = \begin{cases} 1, \text{ if } i\text{-th triangle has been counted} \\ 0, \text{ otherwise} \end{cases}$$

By above definition, $X = m(\sum_{i=1}^{T_3} Z_i)$, and by the linearity of expectation, $E(X) = m(\sum_{i=1}^{T_3} E(Z_i))$. Since the probability of counting a specified triangle is $1/m$, we have $E(Z_i) = 1/m$, and consequently, $E(X) = m \times (T_3 \times 1/m) = T_3$. By definition of the variance,

$$Var(X) = m^2(Var(\sum_{i=1}^{T_3} Z_i) + \sum_{i \neq j} Cov(Z_i, Z_j)).$$

We bound $Cov(Z_i, Z_j)$ by $E(Z_i Z_j)$. $Z_i Z_j$ equals to 1 only when T_i and T_j have a common edge that has been picked by the algorithm. The probability of this event is $1/m$ and since there are at most $d-1$ triangles with a common edge, we have

$$Var(X) \leq m^2(\frac{1}{m}(T_3 + (d-2).T_3)) \leq m(d-1)T_3.$$

Now let Y be the average of $s = 8\frac{1}{\epsilon^2}\frac{md}{T_3}$ parallel instances of X. By Chebyshev's Inequality,

$$Pr(|Y - T_3| \geq \epsilon T_3) < \frac{Var(Y)}{\epsilon^2 E^2(Y)} = \frac{Var(X)/s}{\epsilon^2 {T_3}^2} < \frac{1}{8}.$$

Now for obtaining a (ϵ, δ)-approximation, we run $O(\log \frac{1}{\delta})$ independent estimators, each one succeeding to obtain an ϵ-relative approximation of T with probability at least $\frac{7}{8}$. We need $O(d. \log n)$ space for computing the random variable X and hence, the following result is obtained.

Theorem 1. *For $\epsilon, \delta > 0$, there is a streaming algorithm that outputs an (ϵ, δ)-approximation of T_3, with $T_3 > 0$, using $O(1/\epsilon^2. \log(1/\delta).(\frac{md^2}{T_3}). \log n)$ bit space and per-edge processing time.*

The space usage gets sublinear when $d^2 = o(T_3)$ and close to optimal when d is a constant (see Theorem 3). However the algorithm uses $O(d. \log n)$ bit space for computing the random variable that is poor for graphs with large degree. For our second estimator, we use a random variable that can be computed in $O(\log n)$ bit space.

Second Estimator. To compute the estimator, we need a family of uniform ± 1-valued random vectors of length n, which are 12-wise independent. As indicated in [2], this family can be constructed explicitly using the parity check matrices of BCH codes. These matrices can be constructed with only $O(\log n)$ bits (see [1] for the details). We pick a random vector v from this family (uniformly). Now, as the stream passes, we compute $Z = \sum_{(i,j) \in E} v(i)v(j)$. At the end of stream, we define $X = \frac{1}{6}Z^3$.

We now compute $E(X)$. Based on the definition,

$$E(X) = \frac{1}{6}E((\sum_{(i,j) \in E} v(i)v(j))^3).$$

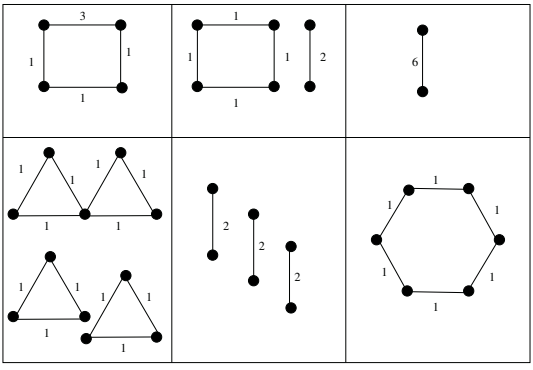


Fig. 1. Subgraphs that increase the variance of the second estimator

Consider that after expanding, each term in the summation corresponds to a specific subgraph of the input graph. By linearity of expectation and regarding the facts that $E(v^{2k+1}(i)) = 0$ and $v(i)$'s are 12-wise independent, the terms that have a variable with an odd power are evaluated to zero. Therefore, only the terms in form of $6v^2(i)v^2(j)v^2(k)$ are remained. These terms correspond to the triangles and thus,

$$E(X) = \frac{1}{6}(6 \times T_3) = T_3.$$

For variance, we have

$$Var(X) = E(X^2) - E^2(X) = \frac{1}{36}(E((\sum_{(i,j)\in E} v(i)v(j))^6) - T_3^2).$$

Similar to the computation of expectation, after expanding, we identify terms which are product of variables with an even power. The different subgraphs that correspond to the terms with an even power are depicted in Fig. 1. Note that the weight of edge (u_i, u_j) in each subgraph equals to the power of $(v(u_i)v(u_j))$ in the corresponding term. For each subgraph, the sum of the weight of edges, incident on each vertex, is even. Therefore according to the figure,

$$Var(X) = \frac{1}{36}(m + 120C_4 + 720{T_3}^2 + 360mC_4 + 720C_6 + 90m^3) - {T_3}^2$$

$$\leq 20({T_3}^2 + mC_4 + C_6 + m^3).$$

Similar to the previous algorithm, we run $s = 160\frac{1}{\epsilon^2}((mC_4 + C_6 + m^3)/{T_3}^2 + 1)$ parallel and independent instances of X and then we take the average of them. Let Y be the average. By Chebyshev's Inequality,

$$Pr(|Y - T_3| \geq \epsilon T_3) < \frac{Var(Y)}{\epsilon^2 E^2(Y)} = \frac{Var(X)/s}{\epsilon^2 {T_3}^2} < \frac{1}{8}.$$

As usual, by taking the median of $O(\log 1/\delta)$ independent and parallel instances of Y, an $(\epsilon, \delta)-$approximation is obtained.

Theorem 2. *For $\epsilon, \delta > 0$, there is a streaming algorithm that outputs an (ϵ, δ)-approximation of T_3, with $T_3 > 0$, using $O(1/\epsilon^2 . \log(1/\delta).(\frac{m^3+mC_4+C_6}{T_3^2})+1). \log n)$ bit space and per-edge processing time.*

2.2 Three-Pass Algorithm

A naive sampling algorithm, that picks samples of vertex triples, should pick at least $O(1/\epsilon^2 . \log(1/\delta).(\frac{T_0+T_1+T_2+T_3}{T_3}))$ random vertex triples. Here, we show how to decrease the number of required samples. The idea is to avoid sampling from triples in Υ_0 and Υ_1 by having the degrees of vertices.

Let d_i be degree of vertex u_i and let $D = \sum_{i=1}^{n} \binom{d_i}{2}$. For picking a random triple, first we pick a vertex randomly while a vertex with degree d_i is picked with probability $\binom{d_i}{2}/D$. Then, from neighbors of the picked vertex, we pick a pair of vertices randomly and uniformly. let a be the picked triple. Since $D = T_2 + 3T_3$, it is easy to see that

$$Pr(a \in \Upsilon_3) = \frac{3T_3}{T_2 + 3T_3}.$$

If we pick the triples based on the above procedure, $O(1/\epsilon^2 . \log(1/\delta).(\frac{T_2+3T_3}{T_3}))$ random triples suffices to $(\epsilon, \delta)-$approximate $|\Upsilon_3|$. Now we show how to implement the above sampling procedure with three passes over the stream. In the first pass, we compute d_i for each u_i. Then we pick the first vertex of the random triples. Let s_i be the number of occurrences of u_i in the sample set. In the second pass, for $i = 1, \ldots, n$, we pick s_i random pairs from neighbors of u_i. The random pairs and the starting vertex form the triples of the sample set. Finally, in the third pass, we determine the number of triangles in the sample set.

Theorem 3. *For $\epsilon, \delta > 0$, there is a streaming algorithm that uses three passes over the stream and produces an (ϵ, δ)-approximation of T_3, with $T_3 > 0$, using $O(n + \frac{1}{\epsilon^2} . \log \frac{1}{\delta} .(\frac{T_2}{T_3} + 1). \log n)$ bit space and per-edge processing time.*

3 Lower Bound

For lower bound, we use reduction to the *Bit-Vector Disjointness* problem in the communication complexity. In this problem, two parties A and B, each one has a binary vector with length of n. A and B want to devise an efficient communication protocol to decide whether their binary vectors are disjoint or not. Our lower bounds are based on a result obtained by Kalyanasundaram and Schnitger [8]. They showed that the length of any communication protocol, succeeding in distinguishing disjoint binary vectors with probability more than $\frac{1}{2}$, must be at least $\Omega(n)$ bits. Since the course of the communication is not restricted, the lower bound also holds for streaming algorithms with constant number of passes.

Theorem 4. *For $\epsilon < 1/3, \delta < 1/2$, every streaming algorithms that output an (ϵ, δ)-approximation of T_3, with $T_3 > 0$, requires at least $\Omega(n/T_3)$ bit space.*

Proof. We represent a pair of binary vectors by an undirected graph so that disjoint binary vectors can be distinguishable via approximating the number of triangles in the graph. Assume T_3 is even and divides n. Let G_1, G_2 and G_3 be n-vertex graphs, such that G_1 and G_2 has $T_3/2$ number of triangles and G_3 with no edges. In each graph, we partition the set of vertices into n/T_3 equal-size parts. For $j = 1, \cdots, n/T_3$, let P_{ij} be the partitions in G_i. Suppose we have an ordering on the set of vertices in each partition. For $k = 1, \cdots, T_3$, let v_{ijk} be the k-th vertex in partition P_{ij}. Consider the binary vectors B_1 and B_2, both with length of n/T_3 . For each j, we add the following edges.

1. For $k = 1, \cdots, T_3$, add an edge between v_{1jk} and v_{2jk}.
2. If $B_1(j) = 1$, for $k = 1, \cdots, T_3$, add an edge between v_{1jk} and v_{3jk}.
3. If $B_2(j) = 1$, For $k = 1, \cdots, T_3$, add an edge between v_{2jk} and v_{3jk}.

Let G be the resulted graph. It is easy to see that if B_1 and B_2 are disjoint, number of triangles in G would be T_3, otherwise number of triangles would be at least $2T_3$. By using an approximation algorithm with relative error less than $1/3$, we can distinguish between graphs with T_3 triangles and ones with at least $2T_3$ triangles. Consequently, we can distinguish disjoint bit-vectors by this algorithm. This completes the proof.

4 Summary

In this paper, we presented three streaming algorithms for counting triangles in graphs with $T_3 > 0$. In the following table, for each algorithm, we have shown when the space usage gets sublinear.

Algorithm	Sublinear space
Naive Sampling	$n^3.\log n/m = o(T_3)$
[3]	$m^{2/3}n.\log n = o(T_3)$
1st Alg.	$d^2.\log n = o(T_3)$
2nd Alg.	$\max\{m, \sqrt{C_4}, C_6/m\}.\log n = o(T_3)$
3rd Alg. (3 passes)	$\max\{n, T_2/T_3\}.\log n = o(T_3)$

However, due to the high dependency of the parameters in the space usage, a clear evaluation and comparison of the algorithms are still unknown to us. We presented a lower bound of $\Omega(n/T_3)$ that only helps in evaluating the first algorithm. A clear lower bound based on the terms T_3 and T_2 will be more useful to evaluate the algorithms. We guess a lower bound of $\Omega(T_2/T_3)$ could exist.

References

1. N. Alon, L. Babai, A. Atai. *A fast and simple randomized algorithm for maximum indepedent set problem.* J. Algorithms 7(1986), 567-583.
2. N. Alon, Y. Matias, M. Szegedy. *The space complexity of approximating the frequency moments.* STOC 96.

3. Z. Bar-Yossef, R. Kumar, S. Sivakumar. *Reduction in streaming algorithms with an application of counting triangles in graphs*. SODA 2002.
4. A.L Buchsbaum, R. Gianvarlo, and J.R Westbrook. *On finding common neighborhoods in massive graphs*. Thoretical Computer Science, 299 (1-3):707-718, 2003.
5. J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, *Graph distances in streaming model; the value of space*. Yale University Technical Report. 2004.
6. J. Feigenbaum, S.Kannan, A. McGregor, S. Suri, and J. Zhang, *On graph problems in a semi-streaming model*. To appear in the 31st International Colloquium on Automata, Languages and Programming, 2004.
7. M. R. Henzinger, P. Raghavan, and S. Rajagopalan, *Computing on data streams*, Technical Report 1998-001, DEC Systems Research Center .1998.
8. B. Kalyanasundaram and G. Schnitger, *The probabilistic communication complexity of set intersection*. SIAM Journal on Discrete Mathematics, 5 545-557, 1990.

A New Approach and Faster Exact Methods for the Maximum Common Subgraph Problem*

W. Henry Suters[1,**], Faisal N. Abu-Khzam[2,**], Yun Zhang[3], Christopher T. Symons[4], Nagiza F. Samatova[4,***], and Michael A. Langston[3,***]

[1] Department of Mathematics and Computer Science, Carson-Newman College CN Box 71958, Jefferson City, TN 37760, USA
[2] Division of Computer Science and Mathematics, Lebanese American University Beirut, Lebanon
[3] Department of Computer Science, University of Tennessee Knoxville, TN 37996–3450, USA
[4] Computer Science and Mathematics Division, Oak Ridge National Laboratory P.O. Box 2008, Oak Ridge, TN 37831–6367, USA

Abstract. The Maximum Common Subgraph (MCS) problem appears in many guises and in a wide variety of applications. The usual goal is to take as inputs two graphs, of order m and n, respectively, and find the largest induced subgraph contained in both of them. MCS is frequently solved by reduction to the problem of finding a maximum clique in the order mn association graph, which is a particular form of product graph built from the inputs. In this paper a new algorithm, termed "clique branching," is proposed that exploits a special structure inherent in the association graph. This structure contains a large number of naturally-ordered cliques that are present in the association graph's complement. A detailed analysis shows that the proposed algorithm requires $O((m+1)^n)$ time, which is a superior worst-case bound to those known for previously-analyzed algorithms in the setting of the MCS problem.

1 Introduction

A popular metric for the similarity of two graphs is the size of their Maximum Common Subgraph (MCS), which is most frequently defined as the largest graph isomorphic to some induced subgraph in each of them. Deciding MCS is $\mathcal{NP}$-complete. It has been studied in bioinformatics [23], chemistry [18, 21], pattern recognition [7, 17], and an assortment of other application areas. A vast literature exists for approximating the size of an MCS. Notably, it is $\mathcal{NP}$-hard to guarantee solutions even within $|V|^{\epsilon}$, where $\epsilon > 0$ and $|V|$ is the size of the MCS [13]. Here we focus on exact MCS algorithms.

* Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy under Contract DE–AC05–00OR22725, by the U.S. National Science Foundation under grant CCR–0311500, by the Office of Naval Research under grant N00014–01–1–0608, and by the U.S. Department of Energy's Genomes to Life program under the ORNL-PNNL project "Exploratory Data Intensive Computing for Complex Biological Systems."
** These two authors contributed equally to this work.
*** Communicating authors: samatovan@ornl.gov, langston@cs.utk.edu

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 717–727, 2005.

These can be roughly classified into three main categories: clique-based methods, non-clique-based backtracking strategies, and various other techniques.

Clique-based methods are the most widely used in the literature. These depend on finding a maximum clique in the association graph [5, 16]. We will define this and other terms in the sequel. Let us just say for now that the association graph is a particular form of product graph built from the two original input graphs. Many clique-based algorithms employ maximal clique enumeration procedures [6, 14, 21], and so are not particularly well suited for maximum clique finding. The general purpose maximum clique algorithm of [22] has a time complexity of $O(1.19^{mn})$, where m and n denote the respective sizes of the input graphs. An example of a non-clique-based backtracking strategy is that of [19], which was developed over two decades ago but rarely used today. In fact it is known to perform well only on graphs of small size [10]. An improved backtracking algorithm has recently been proposed with time complexity $O(m^{n+1}n)$ [15]. Other techniques include algorithms designed for special classes of inputs. One example of this is the dynamic programming approach of [4], which has been proposed for "almost trees of bounded degree." Other methods rely on a set of known graphs against which matching is to be performed [8, 20]. For more information we refer the interested reader to [9, 21].

In this paper, we present and analyze a new clique-based algorithm for the exact MCS problem. To solve clique on the association graph, we actually exploit a special clique structure that we have identified as inherent in the complement of the association graph. There we employ a tree search technique designed to branch on this structure. The resultant "clique branching" algorithm is used to solve vertex cover in the complement and hence clique in the association graph itself. Conceptually, a similar algorithm could be developed to solve the clique problem directly. The two algorithms explore essentially the same search tree. Vertex cover, however, tends to have better reduction rules that allow us to prune the search tree at lower levels and yield faster run times. The algorithm presented here requires $O((m+1)^n)$ time, which (with inputs of course set to ensure that m is at least as large as n) is a better asymptotic worst-case bound than the $O(m^{n+1}n)$ limit of [15] and those known for other previously-analyzed algorithms in the setting of the MCS problem.

2 The Association Graph and Its Properties

All graphs are assumed to be simple, finite, and undirected. Two graphs, G_1 and G_2, are said to be *isomorphic* if there is a one-to-one correspondence between their vertex sets that preserves adjacency. M is a *maximum common subgraph* (MCS) of G_1 and G_2 if M has the largest number of vertices of any graph isomorphic to induced subgraphs of G_1 and G_2. MCS is sometimes referred to a bit more precisely as the *maximum common induced subgraph* (MCIS) problem. Note that an MCS needs to be neither unique nor connected.

Given two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *association graph* $G = (V, E)$ is an undirected graph defined on the vertex set $V = V_1 \times V_2$ with two vertices (u_1, v_1) and (u_2, v_2) being adjacent whenever:

$u_1 \neq u_2$ and $v_1 \neq v_2$, and
either $((u_1, u_2) \in E_1$ and $(v_1, v_2) \in E_2)$ or $((u_1, u_2) \notin E_1$ and $(v_1, v_2) \notin E_2)$.

From this point on, we denote by G_1 and G_2 the two input graphs of MCS. Moreover, we shall assume that $n = |V_1| \leq |V_2| = m$. We know from the definition of the association graph that two vertices (u_1, v_1) and (u_2, v_2) are adjacent in G only if $u_1 \neq u_2$ and $v_1 \neq v_2$. In other words, any two vertices (u_i, v_j), (u_i, v_k) are not adjacent $\forall u_i \in V_1$ and $v_j, v_k \in V_2$, and any two vertices (u_i, v_j), (u_k, v_j) are not adjacent for $\forall v_j \in V_2$ and $u_i, u_k \in V_1$. Therefore, we have the following theorem.

Theorem 1. *If two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are used to create an association graph then, for any $u \in V_1$ and any $v \in V_2$, each of the sets $\{u\} \times V_2$ and $V_1 \times \{v\}$ forms a clique in the complement of the association graph.*

Proof. Let $|V_1| = n$ and $|V_2| = m$. Let us view $V = V_1 \times V_2$ in tabular form as shown.

$V_1 \times V_2$	v_1	v_2	$\cdots$	v_m
u_1	(u_1, v_1)	(u_1, v_2)	$\cdots$	(u_1, v_m)
u_2	(u_2, v_1)	(u_2, v_2)	$\cdots$	(u_2, v_m)
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
u_n	(u_n, v_1)	(u_n, v_2)	$\cdots$	(u_n, v_m)

Then, because of the first two conditions of the definition of the association graph, G, for $\forall u_i \in V_1$, the set of vertices $(u_i, v_1), \ldots, (u_i, v_m)$ in G forms an independent set, and for $\forall v_j \in V_2$, the set of vertices $(u_1, v_j), \ldots, (u_n, v_j)$ in G also forms an independent set. Thus, in the complement graph, each row and column of the table will form a clique. ■

Let k be the size of a common subgraph of G_1 and G_2. We can restrict our search for a minimum vertex cover in the complement of G to covers with sizes that are at least $nm - n$ and at most $nm - k$. Because the maximum value of k is unknown upfront, its lower bound could be the size of any common subgraph that is trivially found (e.g. the graph with the minimum of the sizes of two independent sets of G_1 and G_2 or induced paths of equal size).

Theorem 2. *Let k be the size of any common subgraph of G_1 and G_2, and let G be their association graph. Then any vertex cover of the complement of G must contain at least $nm - \min\{n, m\}$ and at most $nm - k$ vertices.*

Proof. Let H_1 and H_2 be subgraphs of G_1 and G_2 that are isomorphic. If $|H_1| = |H_2| = k$, then the ordered pairs of the set $\{(u, v) : u \in H_1 \text{ and } v \in H_2\}$ form a clique of size k in G. So the complement of G has an independent set of size k. This proves the claim that any vertex cover of the complement of G has at most $mn - k$ vertices.

For the lower bound, we know the size of the MCS is bounded above by $\min\{n, m\}$, which implies the size of the maximum clique of the association graph G does not exceed $\min\{n, m\}$. This implies any vertex cover of the complement of G has at least $(nm - \min\{n, m\})$ vertices. ■

3 A Structural Decomposition Algorithm

In order to motivate our approach it is important to observe that, for a clique of size k, any vertex cover must have at least $k-1$ vertices. Moreover, if any one of the vertices is excluded from the vertex cover, all of its neighbors must be included. In the complement of the association graph, each vertex (u, v) is involved in at least two cliques that only overlap at this vertex. The two cliques correspond to the row and column that intersect at (u, v) in the aforementioned table.

We shall refer to these two cliques by the row-clique and the column-clique of (u, v). If (u, v) is to be excluded from the cover, then all vertices in both of these cliques must be included in the cover. Moreover, any other vertices that are adjacent to this vertex will also be included in the cover. This means the size of the problem is greatly reduced when we decide to exclude a vertex from the vertex cover.

We show that our vertex cover branching algorithm (to follow) can do at least as well as any other known algorithm for MCS. For this purpose, we present vertex cover branching in a way that makes use of the presence of row-cliques and column-cliques in the complement of the association graph.

The idea is that when we attempt to find a vertex cover, we can select at most one vertex from each clique to be excluded from the cover. Thus, in a row-clique of size m, there are $m+1$ possible choices for any vertex cover; m choices each of which excludes one of the vertices, and the remaining choice that includes all the vertices. This forms the basis for our clique branching algorithm.

Theorem 3. *The clique branching algorithm, CliqueBranch(G,n), produces a minimum vertex cover of the complement of the association graph G.*

Proof. We walk through the rows of the aforementioned table, branching at each row. Since a row represents a clique, we could select either to exclude exactly one of its vertices from the vertex cover or to include all of them. We cannot, however, choose a vertex that belongs to a column from which a vertex was selected at a previous branching since each column also represents a clique. The clique branching algorithm examines all possible vertex covers that satisfy these conditions and selects the one with minimum size, thus it produces a minimum vertex cover. ■

In order to get a rough estimate of the complexity of the algorithm, consider the vertex table established in Theorem 1. Branching on the cliques represented by each row, we have two possibilities. If no vertex is excluded from the cover then we have identified m vertices as belonging to the vertex cover. If we select to exclude a particular vertex (u, v), then this vertex will have at least $m + n - 2$ neighbors that must be included in the cover; there are $m - 1$ other vertices in the same row-clique and $n - 1$ other vertices in the same column-clique of (u, v).

We can branch recursively until we arrive at the final row of the matrix. Since there will be n levels of branching, each with at most $m + 1$ possible paths, this produces an algorithm that is at most $O((m + 1)^n)$. This bound is not tight, since once we select a vertex to exclude, its neighbors cannot be selected for a similar role in a later branching. Thus, for subsequent branchings, there will be fewer than $m + 1$ choices.

```
algorithm CliqueBranch(G,n)

Input: the complement of the association graph G created from two graphs
       G1 and G2 of sizes n and m respectively
Output: a minimum vertex cover of the association graph's complement

begin
    MinimumCover = G
    CurrentCover= G
    NumberExcluded = 0
    ExcludeColumns = ∅
    i = 1
    Branch(i)
    output MinimumCover
end

function Branch(i)
    begin
        if i > n then
            if |CurrentCover| < |MinimumCover| then
                MinimumCover = CurrentCover
        else
            NumberExcluded = NumberExcluded + 1
            loop over all v_j ∈ V_2 where j ∉ ExcludedColumns
                if (u_i, v_j) has a neighbor (u_k, v_l) ∉ CurrentCover then
                    do nothing
                else
                    add j to ExcludedColumns
                    remove (u_i, v_j) from CurrentCover
                    Branch (i + 1)
                    remove j from ExcludedColumns
                    add (u_i, v_j) to CurrentCover
            NumberExcluded = NumberExcluded - 1
            Branch (i + 1)
    end
```

Moreover, the number of neighbors of a vertex is more than the number of vertices in its corresponding row and column cliques. This is guaranteed by the following lemma. A formal proof of the computational complexity will be shown in Section 4.

Lemma 1. *Let (u, v) denote a vertex of the complement of the association graph of G_1 and G_2. If the neighborhood of (u, v) is confined to its row-clique and column-clique, then u and v are either both isolated or both connected to all the vertices in each of G_1 and G_2, respectively.*

Proof. Assume the neighborhood condition as stated (and $u' \neq u$ and $v' \neq v$). Then (u, v) is connected to all vertices (u', v') in the association graph of G_1 and G_2. Assume u is neither isolated nor connected to all vertices. Then we can find u' and u"

such that (u, u') is an edge of G_1 while $(u, u")$ is not an edge. If v has an edge (v, v') (or a non-edge $(v, v")$) then (u, v) is not joined to $(u", v')$ (or (u, v) is not joined to $(u', v")$). This is a contradiction. So our assumption about u (neither isolated nor connected to all vertices) is wrong. The same argument proves that if v is not isolated, then it's connected to all vertices of G_2. ■

Thus, if such a pair of vertices (u, v) is found in $G_1 \times G_2$, they are associated together as part of any common subgraph. Moreover, we could detect their presence in the original graphs without searching the association graph. As a pre-processing rule applied prior to any branching step, we could therefore detect the presence of such pairs and reduce the problem size by not including them in the vertex cover.

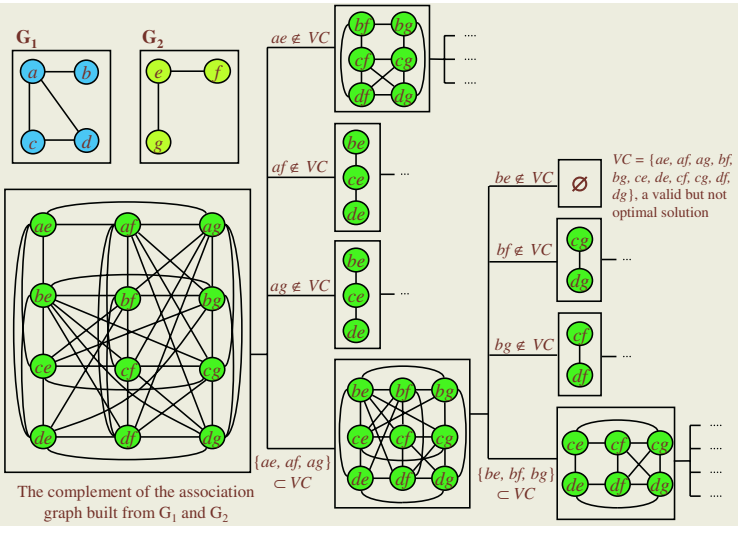


Fig. 1. An example of the clique branching algorithm.

Figure 1 shows an example of the clique branching algorithm applied to two graphs G_1 and G_2, where $|V_1| = 4$ and $|V_2| = 3$. The complement of the association graph has 12 vertices and can be listed as a 4×3 array. Starting from the first row, there are $3 + 1 = 4$ choices (branches). Three of these select a single vertex to be excluded from the cover. The fourth branch does not exclude any vertices. For the first three branches (selecting one), the remaining graphs are much smaller than the original complement graph. This is especially true for the latter two branches. Taking the second branch as an example, excluding vertex af from the cover forces vertices bf, cf, df, in its column, ae, ag, in its row, and bg, cg, dg, in its neighborhood, into the cover. This leaves a graph containing only the three vertices be, ce, de. The next step along this branch will have only two choices: excluding or including vertex be. The fourth branch, which excludes no vertices from the first row, still reduces the graph by including all vertices in the first row in the cover, resulting in a 3×3 array remaining. The next step of this worst-case branch would still have $3 + 1 = 4$ choices.

4 Complexity Analysis

The following theorem is used to determine the complexity of the clique branching algorithm.

Theorem 4. *The complexity of performing the clique branching algorithm on the association graph constructed from two graphs of sizes n and m is*

$$\sum_{i=0}^{n} \frac{m!}{(m-n+i)!} \binom{n}{i}. \tag{1}$$

Proof. Let $R(m,n)$ be the computational complexity of processing an $n \times m$ array of vertices. The first branching will eliminate one row of the array, resulting in $(n-1)$ rows remaining. Also, there are $(m+1)$ possible paths at this branching. There are m paths, each selects a vertex to exclude from the vertex cover, which also eliminates a column from future consideration, resulting in $(m-1)$ remaining columns. The final path does not exclude any vertices from the cover, resulting in m remaining columns. These observations result in the following recurrence relation.

$$R(m,n) = mR(m-1,n-1) + R(m,n-1) \tag{2}$$

with initial condition $R(j,0) = 1$, for $0 \leq j \leq m$.

We next need to demonstrate that the formula

$$R(m,n) = \sum_{i=0}^{k} \frac{m!}{(m-k+i)!} \binom{k}{i} R(m-k+i, n-k), \quad \text{for } 0 \leq k \leq n \tag{3}$$

satisfies the recurrence relation.

We do inductively. First, if $k = 0$ observe that

$$\sum_{i=0}^{k} \frac{m!}{(m-k+i)!} \binom{k}{i} R(m-k+i, n-k) = R(m,n).$$

Next, we need to show that

$$R(m,n) = \sum_{i=0}^{k} \frac{m!}{(m-k+i)!} \binom{k}{i} R(m-k+i, n-k) \tag{4}$$

implies

$$R(m,n) = \sum_{i=0}^{k+1} \frac{m!}{(m-(k+1)+i)!} \binom{k+1}{i} R(m-(k+1)+i, n-(k+1)). \tag{5}$$

Notice that the recurrence relation (Equation 2) implies that

$$R(m-k+i, n-k) = (m-k+i)R(m-k+i-1, n-k-1) + R(m-k+i, n-k-1). \tag{6}$$

Combining Equation 6 with the induction hypothesis (Equation 4), we see that

$$\begin{aligned} R(m,n) &= \sum_{i=0}^{k} \frac{m!}{(m-k+i)!} \binom{k}{i} [(m-k+i)R(m-k+i-1, n-k-1) \\ &\qquad + R(m-k+i, n-k-1)] \\ &= \sum_{i=0}^{k} \frac{m!}{(m-k+i-1)!} \binom{k}{i} R(m-k+i-1, n-k-1) \\ &\quad + \sum_{i=0}^{k} \frac{m!}{(m-k+i)!} \binom{k}{i} R(m-k+i, n-k-1). \end{aligned}$$

As long as $k+1 \leq n$ we define $l = k+1$, re-index the second sum, pull the first term from the first sum and the last term from the second sum to get

$$\begin{aligned} R(m,n) &= \frac{m!}{(m-l)!} R(m-l, n-l) \\ &\quad + \sum_{i=1}^{k-1} \frac{m!}{(m-l+i)!} \left[\binom{l-1}{i} + \binom{l-1}{i-1} \right] R(m-l+i, n-l) \\ &\quad + R(m, n-l). \end{aligned}$$

Since $\binom{l-1}{i} + \binom{l-1}{i-1} = \binom{l}{i}$ we can include the first and last terms to show

$$R(m,n) = \sum_{i=0}^{l} \frac{m!}{(m-l+i)!} \binom{l}{i} R(m-l+i, n-l),$$

which is equivalent to Equation 5. This completes the inductive argument.

We now let $k = n$ to show that

$$R(m,n) = \sum_{i=0}^{n} \frac{m!}{(m-n+i)!} \binom{n}{i} R(m-n+i, 0).$$

Using the initial condition, this shows that

$$R(m,n) = \sum_{i=0}^{n} \frac{m!}{(m-n+i)!} \binom{n}{i}. \blacksquare$$

In general, $\frac{m!}{(m-n+i)!}$ is much smaller than m^{n-i}. Also, there are likely to be fewer than $m+1$ branches at many levels, because some potential branches would result in more neighbors being excluded from the vertex cover. This is guaranteed by Lemma 1. Thus, we expect the performance to be much better than $(m+1)^n$.

5 Remarks

It is well-known that association graphs can be used to reduce MCS to the maximum clique problem. The main contribution of this paper has been a careful analysis of a clique-branching algorithm designed to exploit the structure implicit in the complement of the association graph. The results presented here are highly theoretical, and provide only an asymptotic upper bound on run times. Experimental comparisons with other methods are now underway in order to gauge expected performance. It is not clear whether direct maximum clique algorithms that operate on the association graph itself can do better than our clique-branching approach. We do know, however, that any such algorithm should not do worse than the time bound we derive in Equation 1, at least as long as it performs a standard form of vertex branching. To illustrate, consider recursive backtracking algorithms such as that of [22]. The following branching method is employed: at each node of the search tree, a highest-degree vertex v is selected and two possible choices are explored: either v is in the clique or it is not. When v is added to the clique, all vertices not adjacent to it are eliminated. Thus, the problem size may be reduced considerably in this case. When v is not added to the clique, v alone can of course be deleted.

In the association graph, a vertex $(u, v) \in V$ where $u \in V_1, v \in V_2$ is a member of an independent set containing all vertices whose first component is u. It is also a member of another independent set consisting of all vertices whose second component is v. Therefore, when using a maximum-clique algorithm, if (u, v) is added to a (potential maximum) clique, then all such vertices are deleted. So the following recursive equation holds for the run time $T(nm)$:

$$T(nm) = T((n-1)(m-1)) + T(nm-1).$$

And by Lemma 1, we know the equation can be made better since there must be other neighbors of the vertex (u, v). The second term of the above equation can be expended as follows:

$$T(nm-1) \leq T((m-1)(n-1)) + T(nm-2).$$

Combining the above two equations, we get

$$T(nm) \leq 2T((n-1)(m-1)) + T(nm-2),$$

which leads to

$$T(nm) \leq mT((n-1)(m-1)) + T(nm-m).$$

This is equivalent to equation 2. Thus, maximum clique algorithms could achieve performance similar to that of our clique branching method. If we were to analyze (blindly) the run time of maximum-clique methods, without accounting for the number of vertices that are eliminated at each branching step, the best known algorithm would be assumed to take a running time of $O(2^{\frac{nm}{4}})$ [22].

We often solve maximum clique by reducing it to the minimum vertex cover problem, simply because a clique in a graph is the complement of a vertex cover in its

complement. Recent efficient vertex cover algorithms based on the theory of fixed-parameter tractability [11] have proved very useful, especially when the size of the clique is large [2]. Such algorithms target the parameterized version of a problem. A natural parameter that we could associate with the input of MCS is the size of the common subgraph. In other words, the parameterized MCS problem (k-MCS) can be posed as follows:

> Given: A pair of graphs, G_1 and G_2, and a positive integer k.
> Question: Do G_1 and G_2 have a common (induced) subgraph whose order is at least k?

Let G be the association graph of G_1 and G_2. The search for a common subgraph of size k (or more) is equivalent to the search for a k-clique in G. Thus, as noted earlier, we look for a vertex cover of size $nm - k$ in the complement of G. Since k is not larger than n, the size of the sought cover is bounded below by $nm - \sqrt{nm}$, which is huge when compared to nm. So the use of fixed-parameter vertex cover algorithms may seem not feasible for k-MCS. This is also supported by the fact that MCS is W[1]-hard [11]. Nevertheless, our algorithm is a straightforward branching approach that achieves the best current running time for MCS. The advantage of using parameterized vertex cover algorithms (rather than direct clique algorithms) is mainly due to two observations. First, vertex cover branching uses the same universal strategy: if a vertex is not in the cover, then all its neighbors must be in the cover. So the two algorithms explore essentially the same search space. Second, when the degree of each vertex in the complement graph drops below a certain constant c (due to the continual removal of vertices during branching), the resulting graph must have a vertex cover whose size is smaller than $\frac{c-1}{c}$ of the resulting graph size. (This being true since a graph of maximum degree c has an independent set of size at least $1/c$ of the graph size). Thus, parameterized vertex cover techniques, such as preprocessing and kernelization [1, 3], may be applied together with branching to reduce the size of the search space and produce better run times. Finally, for completeness it is probably worth pointing out that, if one is dealing with labeled graphs, then MCS is potentially an easier problem. Simpler maximum clique algorithms are often used on labeled and other restricted types of association graphs [7, 12] to achieve better performance.

References

1. F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proceedings, Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2004.
2. F. N. Abu-Khzam, M. A. Langston, and P. Shanbhag. Scalable parallel algorithms for difficult combinatorial problems: A case study in optimization. In *Proceedings, International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pages 563-568, 2003.
3. F. N. Abu-Khzam, M. A. Langston, and W. H. Suters. Effective vertex cover kernelization: A tale of two algorithms. In *Proceedings, ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*, 2005.

4. T. Akutsu. A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. *IEICE Trans. Fundamentals*, E76-A:1488-1493, 1993.
5. I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. In D. Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4. Boston MA: Kluwer Academic Publishers, 1999.
6. C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, 16:575-577, 1973.
7. H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In *Proc. IAPR Workshop on Structural and Syntactic Pattern Recognition*, 2002.
8. K. Shearer H. Bunke and S. Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition*, 34:1075-1091, 2001.
9. D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265-298, 2004.
10. D. Conte, C. Guidobaldi, and C. Sansone. A comparison of three maximum common subgraph algorithms on a large database of labeled graphs. In E. Hancock and M. Vento, editors, *IAPR Workshop GbRPR 2003, LNCS 2726*, pages 130-141, 2003.
11. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.
12. P.J. Durand, R. Pasari, J.W. Baker, and Chun che Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2, 1999.
13. V. Kann. On the approximability of the maximum common subgraph problem. In *STACS '92: Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, pages 377-388. Springer-Verlag, 1992.
14. I. Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250:1-30, 2001.
15. E. B. Krissinel and K. Henrick. Common subgraph isomorphism detection by backtracking search. *Software Practice and Experience*, 34:591-607, 2004.
16. G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9:341-352, 1972.
17. A. Massaro and M. Pelillo. Matching graphs by pivoting. *Pattern Recognition Letters*, 24(8):1099-1106, 2003.
18. J. McGregor and P.Willett. Use of a maximal common subgraph algorithm in the automatic identification of the ostensible bond changes occurring in chemical reactions. *Journal of Chemical Information and Computer Science*, 21:137-140, 1981.
19. J. J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software Practice and Experience*, 12:23-34, 1982.
20. B. T. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs*. Phd, University of Bern, 1995.
21. J.W. Raymond and P.Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16:521-533, 2002.
22. J. M. Robson. Finding a maximum independent set in time $O(2n/4)$. Technical Report 1251-01, Universite Bordeaux I, LaBRI, 2001.
23. A. Yamaguchi, K. F. Aoki, and H. Mamitsuka. Finding the maximum common subgraph of a partial k-tree and a graph with a polynomially bounded number of spanning trees. *Information Processing Letters*, 92(2):57-63, 2004.

On the Power of Lookahead in On-Line Vehicle Routing Problems

Extended Abstract

Luca Allulli, Giorgio Ausiello*, and Luigi Laura

Dip. di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria, 113 - 00198 Roma Italy
{allulli,ausiello,laura}@dis.uniroma1.it

Abstract. Vehicle Routing Problems are generalizations of the well known Traveling Salesman Problem; we focus on the *on-line* version of these problems, where requests are not known in advance and arrive over time. We introduce a model of lookeahead for this class of problems, the *time lookahead* Δ, which allows an on-line algorithm to foresee all the requests that will be released during next Δ time units. We present lower and upper bounds on the competitive ratio of known and studied variants of the OlTsp; we compare these results with the ones from the literature. Our results show that the effectiveness of lookahead varies significantly as we consider different problems.

1 Introduction

In the classical Traveling Salesman Problem (Tsp) the goal is to visit a set of cities minimizing the traveled distance [10]. In this paper we deal with *on-line* variants of Tsp, in which each request comes with a *release time*, and cannot be served before it. These problems are collectively known as *On-line Vehicle Routing Problems* (OlVrps).

OlVrps model many real-world problems. For such problems an *on-line algorithm*, that becomes aware of the requests over time, is required to serve such requests at its best, although, clearly, in such a situation of uncertainty the optimal solution cannot in general be achieved. In order to evaluate the quality of the solution produced by an on-line algorithm we use competitive analysis. *Competitive analysis* was formally introduced by Sleator and Tarjan in [17], though the underlying idea was already used by Graham [8]. We say that an on-line algorithm A is ρ-competitive ($\rho \in \mathbb{R}^+$) if, for any input instance σ, $\mathsf{A}(\sigma) \leq \rho \cdot \mathsf{OPT}(\sigma)$; we denote by $\mathsf{A}(\sigma)$ and $\mathsf{OPT}(\sigma)$ the cost, on input σ, of the solution found by A and of the optimal solution, respectively. See also [6] for an overview of competitive analysis and on-line algorithms.

Many OlVrps have been studied using competitive analysis. Yet, competitive analysis has been criticized for being too strict, since it is often possible

* Work partially supported by the MIUR-PRIN ALGO-NEXT project.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 728–736, 2005.

to build up pathological input instances that only an off-line server (intuitively called "adversary") can serve effectively, thanks to its clairvoyance. In order to limit, in some way, the power of the off-line adversary, restricted types of adversary have been proposed (see [5, 12, 13]).

In this paper we explore the influence of lookahead on several OlVrps. It is interesting, from both theoretical and practical points of view, to evaluate the improvement that we can achieve, in terms of competitiveness, if we allow on-line algorithms to foresee some future requests. The influence of lookahead on the competitive ratio of on-line algorithms was first studied by Chung *et al.* for the dynamic location problem [7], and then analyzed in many other problems, among which we cite scheduling, paging, list-update, bin packing and various graph problems [1, 2, 9, 11, 16]. Concerning OlVrps, we define two kinds of lookahead: time lookahead and request lookahead. An online algorithm has *time lookahead* Δ if it is able to foresee all requests that will be released during the next Δ time units; it has *request lookahead* k if it can foresee the next k requests that will be released. The scope of our investigation is twofold. On the one hand, lookahead is a feature that, to some extent, can be easily implemented in real-world applications of routing algorithms; for example, a radio-taxi service could require customers to book rides in advance. Furthermore, lookahead can be used to limit the power of the adversary in a natural and parametric way, since it compels the adversary to disclose in advance to the on-line algorithm some of the future requests. In this paper we deal with time lookahead; some results about request lookahead can be found in [3].

Our results show that the effectiveness of lookahead varies significantly as we consider different variants of the problems, and only in some cases it is worthy to adopt it (see also [15]). For the classical HTsp, that is the problem in which the server is forced to return to the origin at the end of the tour, we found a lower bound of 2 and a matching upper bound, but these results hold also without lookahead (see [4]). If we consider the Nomadic version of the problem, in which the costraint of ending the tour in the origin is dropped, lookahead proves to be usuful, and we can show better bounds than the ones in [14]. Moreover, if we focus on problems where the underlying metric space is the (limited) real line, we see that lookahead and the bounds are reciprocally related, and the competitiveness tends to 1 when lookahead increases. We also considered the Net latency Tsp (NlTrp), where the objective function is the sum of the serving time of each request minus its release time, for which no competitive algorithm exists. Indeed for this problem no algorithm is competitive even if it is provided with a quite large amount of time lookahead (namely $\Delta \in [0, 2D[$, being D the diameter of the metric space), in any metric space; moreover, no competitive algorithm exists at all in metric spaces that are isomorphic to open sets of $\mathbb{R}^d$, $d \geq 2$.

The remainder of this paper is organized as follows. Next section introduces the basic definitions. In Section 3 we show lower bounds on the competitive ratio of algorithms endowed with lookahead, while in Section 4 we present algorithms that successfully exploit lookahead. Section 5 presents our conclusions.

2 Preliminaries

In an instance of an Online Traveling Salesman Problem we are given a metric space $M = (X, d)$, where X is a set of points and d is a distance function on X, with a distinguished point $O \in X$, the *origin*; and a set of requests $\sigma = \{\sigma_1, \ldots, \sigma_n\}$. Each request consists of a pair $\sigma_i = (x_i, t_i) \in X \times \mathbb{R}_0^+$, where x_i is the *position* of σ_i, and t_i is its release time. A *server* is located in the origin at time 0, and thereafter moves in the metric space, at most at unit speed, in order to *serve* all the requests, i.e. to visit each point x_i where a request is placed, not earlier than the release time t_i of the request. The additional constraint can be required that the server return to the origin after having served all the requests. The goal of the server is to find a feasible schedule that minimizes an *objective function*, which in some way measures the quality of the schedule. Many objective functions have been proposed in literature. In this paper we deal with some of the most important ones: the *completion time*, i.e. the time when the server completes its service; the *net latency*, i.e. the sum of the times each request has to wait to be served minus its release time, namely $\sum_{i=0}^{n}(\tau_i - t_i)$, where τ_i is the time instant when request σ_i is served; and the *latency*, i.e. the sum of the serving times for each request, $\sum_{i=0}^{n} \tau_i$. The variant where we aim at minimizing the completion time with the additional constraint that the server return to the origin after the service is referred to as the *Homing Traveling Salesman Problem* (HTSP); if we do not require the additional constraint, then we get the *Nomadic Traveling Salesman Problem*. If the server has to minimize the net latency we deal with the *Net Latency Traveling Repairman Problem* (NLTRP); the last variant we consider is the *Latency Traveling Repairman Problem* (LTRP), where the objective function is the latency.

3 Lower Bounds

In this section we give lower bounds on the competitive ratio of algorithms with lookahead for OLVRPS.

The first theorem shows a general lower bound of 2 for both the HTSP and the NTSP, for any value of time lookahead. It extends the validity of the lower bound of 2 for the HTSP without lookahead, first presented by Ausiello *et al.* [4], and later alternatively proved by Lipmann [14]. We draw inspiration from the proof by Lipmann. Notice that Ausiello *et al.* [4] also presented a 2-competitive algorithm for the HTSP without lookahead: this means that lookahead cannot improve the competitive ratio of algorithms for the HTSP in general metric spaces.

Theorem 1. *No deterministic algorithm for the* HTSP *or the* NTSP *can achieve a competitive ratio better than 2, even when time lookahead is provided.*

Proof. Consider a *spider graph* $G = (V, E)$ with $N + 1$ nodes: a central node v_0 and N peripheral nodes $v_1, \ldots, v_N$. Each peripheral node v_i is connected to the

central node by an edge $e_i = \{v_0, v_i\}$ having length 1/2. Let A be any algorithm for the HTSP or the NTSP on G with time lookahead Δ.

At time Δ, N requests are presented, one in each peripheral node. Let $t_{stop} = \Delta + N - 1$. For any time $t \leq t_{stop}$, if A serves one request in vertex v_i at time t, then a new request is presented in the same vertex v_i at time $t + \Delta$; A can see it immediately according to its lookahead. Thus, at any time $t \leq t_{stop}$, A is aware of exactly N requests that either have been released but not served or will be released in the future. In particular, at time t_{stop}, A must still serve N requests, and cannot finish before time $t_{stop} + N - 1 = \Delta + 2N - 2$.

On the other hand, an off-line adversary can complete its service not later than time $2\Delta + N$. In fact it can serve requests in the following order: first the requests in vertices that are not touched by A before time t_{stop}, if any; then, all the other requests, visiting peripheral vertices once and in the same order A visits them for the last time. This way, giving to the off-line adversary a delay of at least Δ over A, it can serve the newly presented requests in every peripheral vertex along with the old ones. Thus the adversary finishes not later than time $(1/2 + 2\Delta) + (N - 1) + 1/2$: the first term is a time sufficient to reach the first request and to gain a delay of Δ over A; the second term is a time sufficient to serve all the requests, and the last term is a time sufficient to return home, if the problem is the HTSP.

The lower bound on the competitive ratio

$$\frac{\mathsf{A}(\sigma)}{\mathsf{OPT}(\sigma)} \geq \frac{2N + \Delta - 2}{N + 2\Delta}$$

can be rendered arbitrarily close to 2 by choosing a sufficiently large value for N.

Let us now consider the objective function of net latency, for which no competitive algorithm without lookahead exists. The following theorem shows that, in general, time lookahead does not help.

Theorem 2. *Let Ω be a open set of $\mathbb{R}^k$, $k \geq 2$; let* A *be an online algorithm for the* NLTRP *on Ω with time lookahead Δ. Then, for all $\Delta \in \mathbb{R}^+$,* A *is not competitive for the* NLTRP *in Ω.*

Proof. (Sketch) We will refer to the online algorithm as A, to the adversary as B. Without loss of generality we suppose that $\Omega \subseteq \mathbb{R}^2$.

We construct a grid G of $2N$ points such that, in order to visit any subset of G with N points, a minimum time of Δ is needed. The adversary releases some *starting requests*, consisting of at least one request in each point of G. Furthermore, the adversary selects a subset $G^- \subset G$ containig N points, and forces A to serve requests in G^- first, in such a way that otherwise A cannot be competitive. This is achieved by suitably tuning the number of requests released in each point of G. While A serves the starting requests in G^-, B serves all the other starting requests; afterwards, B begins to follow A with a delay of Δ. In the meanwhile, new requests are generated: if A serves some requests at time t, then B releases a new request in the same point at time $t + \Delta$. B is able to

serve each new request on the fly, at no cost. On the other hand, at any time t there are always requests in $2N$ points that either have been released but not served by A, or will be released soon, not later than time $t + \Delta$. During the next Δ time units A will be able to serve requests in at most N of these $2N$ points. Consequently it will be late on at least N requests, paying some cost for serving them. By iterating this procedure, B can force A to pay an arbitrarily large cost. Since B pays only a fixed cost in order to serve the starting requests, we get the lower bound.

The technique we used to prove the last theorem is based on the bi-dimensional density of $\mathbb{R}^2$, which makes it possible to force the on-line algorithm to take an arbitrarily long tour in order to serve all requests, independently from the diameter of the metric space. In other kinds of metric spaces, such as uni-dimensional or discrete spaces, the same technique cannot be used. Anyway, we now show that no algorithm can be competitive in any metric space, even with time lookahead Δ, if Δ is less than two times the diameter of the metric space. This is still quite a large amount of lookahead for many real world applications.

Theorem 3. *Let $M = (X, d)$ be any metric space with diameter D, and* A *any algorithm for the* NLTRP *on M with time lookahead Δ. If $\Delta < 2D$, then* A *is not competitive.*

Proof. (Sketch) We denote by B the off-line adversary. B releases requests in two points, P and Q, that are "far enough": think, for example, of two points whose distance is exactly D, if they exist. At the beginning, only two requests are presented, one in P and one in Q. Every time t A serves a request, a new request is released in the same point at time $t + \Delta$. Then A can always see one request in P and one request in Q, either released but not served or that will be released soon. Assume, without loss of generality, that A is located in P at time t. If A moves to Q, serves the other request and comes back to P, the new request in P will be released before A comes back, because $2D > \Delta$: then A will serve the new request late. Otherwise, if A waits in P for the new request to be released, it will serve the pending request in Q late. In any cases, A will have to pay some finite cost. By iterating this trick a suitable number of times, B will charge A an arbitrarily large cost.

On the other hand, B at the beginning serves the request not served by A as the first one, and afterwards follows A with an exact delay[1] of Δ; this delay allows B to serve all the newly released requests on the fly. Thus, B pays only a finite cost for serving the first two requests.

4 Algorithms

In this section we describe algorithms which make use of time lookahead and have a lower competitive ratio than their counterparts without lookahead.

[1] We need the technical assumption $\Delta > D$, which obviously does not alter the generality of our theorem.

We begin with the NTSP. For this problem, in the general limited metric space we match the lower bound of 2 (Theorem 1) using a time lookahead of $\Delta = D$, where D is the diameter of the metric space. More specifically, when Δ varies from 0 to D, the competitive ratio continuously decreases from $1 + \sqrt{2}$ (the current best upper bound without lookahead [14]) and 2. Our algorithm is a natural extension of the algorithm $\mathsf{ReturnHome}_\alpha$, presented in [14].

Algorithm 4 ($\mathsf{ReturnHome}_\alpha$ with time lookahead Δ) *At every time $t \in \mathbb{R}^+$, algorithm $\mathsf{ReturnHome}_\alpha$ (RH_α) either is idle or is following a tour T. RH_α is a parametric algorithm, with parameter $\alpha \in]0, 1]$. If at time t RH_α is following a tour T, then it stays within a distance of αt from the origin (*ball-constraint*). In particular, RH_α moves at full speed as long as the ball-constraint is satisfied; otherwise it regulates its speed to the maximum amount while still respecting the ball-constraint.*

Initially, RH_α is idle. Independently of its current state, as soon as RH_α foresees a new request according to its lookahead, it immediately returns to the origin, and waits for the new request to be actually released. Then, it begins to follow the minimum-length tour T over all the released but not yet served requests.

Theorem 5. $\mathsf{ReturnHome}_\alpha$ *with lookahead $\Delta = \delta D$ is a ρ_α-competitive algorithm for the* NTSP *in any metric space with diameter D, where*

$$\rho_\alpha = \max\left\{\frac{1}{\alpha},\, 2 + \alpha\frac{1-\delta}{1+\alpha\delta}\right\}.$$

This value is minimized when $\alpha = \frac{\delta - 2 + \sqrt{\delta^2+8}}{2(\delta+1)}$.

Proof. We distinguish two cases, depending on whether the server has to regulate its speed during the last tour, i.e. the tour scheduled after the last request is released.

Case 1. The server must regulate its speed during the last tour. Let $\sigma_r = (x_r, t_r)$ be the last request that forces RH_α to regulate its speed. Since RH_α regulates its speed only when it is necessary, σ_r is served at time $\frac{x_r}{\alpha}$. Thereafter, RH_α completes the optimum tour at full speed: let $|T_r|$ be the length of the remaining part of the tour. We have that $\mathsf{RH}_\alpha(\sigma) = \frac{x_r}{\alpha} + |T_r| \leq \frac{1}{\alpha}(x_r + |T_r|)$. On the other hand, the off-line adversary pays at least $x_r + |T_r|$ to get to x_r and serve the remaining requests. Therefore: $\frac{\mathsf{RH}_\alpha(\sigma)}{\mathsf{OPT}(\sigma)} \leq \frac{1}{\alpha}$.

Case 2. The server does not regulate its speed during the last tour. Assume that the last request is released at time $t + \delta D$; it is foreseen by RH_α at time t, thanks to its lookahead. RH_α will immediately come back to the origin, and will start following the optimal tour T at the first time $t_{start} \geq t + \delta D$ when the server is in the origin. The completion time of RH_α is thus $\mathsf{RH}_\alpha(\sigma) = t_{start} + |T|$. Now we give two lower bounds on the optimal cost. Since the off-line adversary must visit all the requests, it pays at least $|T|$. Since it must serve the

last request not earlier than its release time, it pays at least $t + \delta D$. Hence, the competitive ratio of RH_α can be bounded by:

$$\frac{\mathsf{RH}_\alpha(\sigma)}{\mathsf{OPT}(\sigma)} = \frac{t_{start} + |T|}{\mathsf{OPT}(\sigma)} \leq \frac{t_{start}}{t + \delta D} + \frac{|T|}{|T|} = \frac{t_{start}}{t + \delta D} + 1.$$

In the following, we show that $\frac{t_{start}}{t+\delta D} \leq 1 + \alpha\frac{1-\delta}{1+\alpha\delta}$. We distinguish three subcases, considering how much time has elapsed since time 0. Intuitively, if few time has elapsed, then the server is near the origin and can quickly return home; if much time has elapsed, then t_{start} is comparable to $t + \delta$, because the metric space is limited and RH_α cannot be "too far" from the origin. The worst subcase is in the middle.

- If $\alpha t \leq \delta D$, then RH_α is near the origin so that it can return home not later than time $t + \delta D$, when the request is released. Thus $t_{start} = t + \delta D$, and so $\frac{t_{start}}{t+\delta D} = 1$.
- If $\delta D < \alpha t \leq D$, then $t_{start} \leq t + \alpha t$. Thus $\frac{t_{start}}{t+\delta D} \leq \frac{t+\alpha t}{t+\delta D}$; it is easy to see that the latter quantity is monotonically increasing in t when $\alpha t \in]\delta D,\, D]$; hence it reaches the maximum value of $1 + \alpha\frac{1-\delta}{1+\alpha\delta}$ when $\alpha t = D$.
- If $\alpha t > D$, we use the fact that $t_{start} \leq t + D$, because the metric space has diameter D. Thus $\frac{t_{start}}{t+\delta D} \leq \frac{t+D}{t+\delta D}$; it is easy to see that the latter quantity is monothonically decreasing in t when $\alpha t \in [D, +\infty[$. As before, it reaches the maximum value of $1 + \alpha\frac{1-\delta}{1+\alpha\delta}$ when $\alpha t = D$.

From Case 1 and Case 2 we infer that

$$\frac{\mathsf{RH}_\alpha(\sigma)}{\mathsf{OPT}(\sigma)} \leq \max\left\{\frac{1}{\alpha},\, 2 + \alpha\frac{1-\delta}{1+\alpha\delta}\right\}.$$

It is easy to see that, for any fixed value of $\delta \in [0, 1]$, this quantity is minimized by choosing $\alpha = \frac{\delta - 2 + \sqrt{\delta^2+8}}{2(\delta+1)}$.

For the metric space of the limited line (i.e. the segment), better algorithms exist. In the following we present an algorithm for the NTSP whose competitive ratio tends to 1 when Δ increases.

Algorithm 6 (Optimize Earlier Requests Only) *At time* 0*, algorithm* Optimize Earlier Requests Only *(*OERO*) foresees all the requests that will be released up to time Δ. It computes the optimal schedule over these requests, and begins to follow it. After time Δ, if new requests are released, then* Optimize Earlier Requests Only *switches to another mode, even if it has not completed the scheduled tour. It begins to sweep the segment backwards and forwards, serving all the requests it encounters.*

Theorem 7. Optimize Earlier Requests Only *with time lookahead $\Delta = \delta D$ is a $(1 + 2/\delta)$-competitive algorithm for the* NTSP *defined on a segment with length D.*

Proof. Let σ be the input instance. If no requests are released after time Δ, OERO is clearly 1-competitive. Otherwise, we have that $\mathsf{OPT}(\sigma) \geq \delta D$. Let $\sigma^* = (x^*, t^*)$ be the last request served by OERO. Since after time Δ an active request waits at most $2D$ time units before being served, we have that $\mathsf{OERO}(\sigma) \leq t^* + 2D$. Obviously $\mathsf{OPT}(\sigma) \geq t^*$. Then we obtain $\mathsf{OERO}(\sigma) \leq \mathsf{OPT}(\sigma) + (2/\delta)\mathsf{OPT}(\sigma) = (1 + 2/\delta)\mathsf{OPT}(\sigma)$.

OERO can be modified in order to work for the HTSP and the LTRP, yielding the same competitive ratio. The proof in the case of the HTSP is very similar to that for the NTSP; in the case of the LTRP the proof is slightly different because it is necessary to sum the contributions of the single requests. In this extended abstract we omit both these proofs; they can be found in [3].

5 Conclusions

In this paper we defined a model of lookahead for OLVRPS. We showed that the effectiveness of lookahead is closely related to the specific problem considered, thus providing a new perspective on the problem itself. Nevertheless, intuition suggests that it should always be possible to improve the empirical performance of algorithms for OLVRPS using lookahead; an experimental evidence confirming such intuition is presented in [3].

It is noteworthy to mention that, in general, lookahead makes more difficult the task of providing lower bounds. Usually, looking at the competitive analysis as a game between the on-line player and an evil adversary, the latter can place request because it knows which future decisions will be taken by the on-line algorithm; if we add lookahead to the game, these decisions depend, in turn, on the requests: we have a mutual dependency that makes the analysis task more complex.

Finally, it would be interesting to discover whether competitive algorithms with lookahead exist for two prominent objective functions: the net latency in uni-dimensional spaces when $\Delta \geq 2D$, and the maximum flow time (see [13]).

References

1. S. Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, 1997.
2. S. Albers. A competitive analysis of the list update problem with lookahead. *Theor. Comput. Sci.*, 197(1-2):95–109, 1998.
3. L. Allulli, G. Ausiello, and L. Laura. On the power of lookahead in on-line vehicle routing problems. TR-02-05, DIS, Università di Roma "La Sapienza".
4. G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 2001.
5. M. Blom, S. O. Krumke, W. E. de Paepe, and L. Stougie. The online-TSP against fair adversaries. *INFORMS Journal on Computing*, 13:138–148, 2001.
6. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

7. F. R. K. Chung, R. L. Graham, and M. E. Saks. A dynamic location problem for graphs. *Combinatorica*, 9(2):111–131, 1989.
8. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
9. E. F. Grove. Online bin packing with lookahead. In *SODA '95: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 430–436. Society for Industrial and Applied Mathematics, 1995.
10. G. Gutin and A. P. Punnen, editors. *The Traveling Salesman Problem and its Variations*. Kluwer, Dordrecht, The Nederlands, 2002.
11. S. Irani. Coloring inductive graphs on-line. *Algorithmica*, 11(1):53–72, 1994.
12. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance [scheduling problems]. In *FOCS '95: Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, page 214. IEEE Computer Society, 1995.
13. S. O. Krumke, L. Laura, M. Lipmann, A. Marchetti-Spaccamela, W. E. de Paepe, D. Poensgen, and L. Stougie. Non-abusiveness helps: an $O(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem. In *APPROX*, pages 200–214, 2002.
14. M. Lipmann. *On-Line Routing*. PhD thesis, Technical University of Eindhoven, 2003.
15. M. Lipmann, X. Lu, W. de Paepe, R. Sitters, and L. Stougie. On-line dial-a-ride problems under a restricted information model. In *Proc. 10th European Symp. on Algorithms (ESA)*, pages 674–685, 2002.
16. R. Motwani, V. Saraswat, and E. Torng. Online scheduling with lookahead: Multipass assembly lines. *INFORMS J. on Computing*, 10(3):331–340, 1998.
17. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

Efficient Algorithms for Simplifying Flow Networks*

Ewa Misiołek[1] and Danny Z. Chen[2]

[1] Mathematics Department, Saint Mary's College, Notre Dame, IN 46556, USA
[2] Department of Computer Science and Engineering, University of Notre Dame
Notre Dame, IN 46556, USA

Abstract. Computing flows in a network is a fundamental graph theory problem with numerous applications. In this paper, we present two algorithms for simplifying a flow network $G = (V, E)$, i.e., detecting and removing from G all edges (and vertices) that have no impact on any source-to-sink flow in G. Such network simplification can reduce the size of the network and hence the amount of computation performed by maximum flow algorithms. For the undirected network case, we present the first linear time algorithm. For the directed network case, we present an $O(|E| * (|V| + |E|))$ time algorithm, an improvement over the previous best $O(|V| + |E|^2 \log |V|)$ time solution. Both of our algorithms are quite simple.

1 Introduction

Computing flows in a network $G = (V, E)$ (i.e., a graph) is one of the most fundamental problems in graph theory and has numerous applications in the real world. Its goal is to find a flow with certain specific characteristics that can travel from the starting node (source) to the destination node (sink) along the edges of the network under specified constraints, e.g., limited edge capacity, cost, etc. Flow problems are of great importance in areas such as computer networks, computer vision, combinatorial optimization, and transportation. Many other key theoretical problems, such as minimum cut and bipartite matching, can be formulated as network flow problems. For an extensive survey of flow problems, algorithms, and their applications, see [2, 8, 9, 12].

Because of the wide-range applications of flow problems, it is important to make the algorithms for them run as fast as possible. Since flow networks can be of a very large size, and the time bounds of most flow algorithms are polynomials of rather "high" degrees (i.e., super-linear in both $|E|$ and $|V|$), it is highly desirable to reduce, as much as possible, the size of the input network by removing some or even all edges (and vertices) without affecting the resulting flow. In this paper, we consider the problem of reducing the size of a flow network $G = (V, E)$

* This research was supported in part by the National Science Foundation under Grant CCR-9988468 and by a Summer Graduate Research Fellowship of the Center for Applied Mathematics of the University of Notre Dame.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 737–746, 2005.

(a) (b)

Fig. 1. (a) The undirected edges on the cycle C are useless. (b) The directed edge e is useless.

by detecting and removing all edges (and vertices) that do not contribute to any source-to-sink flow under all possible edge capacity assignments in G. This is called the flow network simplification problem. The edges removed by such a simplification are called *useless edges* in G.

Definition 1. *Given a network $G = (V, E, s, t)$, where s is the source vertex and t is the sink vertex, an edge $e \in E$ is said to be* useful *if there exists an assignment of edge capacities in G such that an s-to-t flow uses e. Any edge of G that is not useful is said to be* useless.

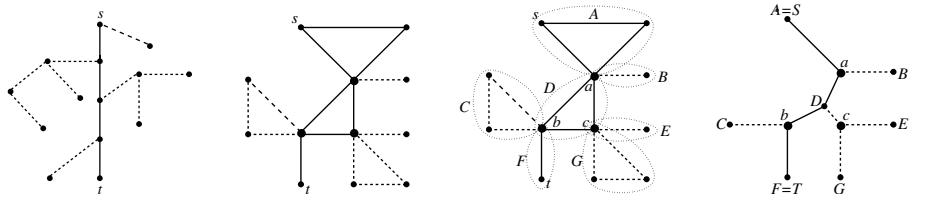
Such a network $G = (V, E, s, t)$ is called a *flow network* (or *flow graph*). By the above definition, useless edges are those that cannot be used by any s-to-t flow under all possible assignments of edge capacities in G. Some obvious useless edges are those that are unreachable from the source s and those that cannot reach the sink t. For example, consider the two networks in Figure 1. Note that all edges on the cycle C in Figure 1(a) are useless since any s-to-t flow through these edges is "blocked" at the vertex x. These edges thus cannot contribute to any s-to-t flow. Similarly, the edge e in the directed network in Figure 1(b) is useless.

Hence, the flow network simplification problem can be defined as follows: Given a flow network $G = (V, E, s, t)$, where s is the source vertex and t is the sink vertex, remove all useless edges in G.

To the best of our knowledge, the only result on this problem was due to Biedl, Brejová, and Vinař [4]. The authors considered only directed networks and presented two algorithms, one for planar directed graphs and the other for general directed graphs. On the case of general directed networks, they proved that the problem of removing all useless edges is NP-complete. After relaxing the definition of useless edges, they gave an $O(|V| + |E|^2 \log |V|)$ time algorithm for removing only some of the useless edges. For the planar directed network case, they provided an $O(|V|^2)$ time algorithm for removing all useless edges.

We are not aware of any published result on the simplification of undirected flow networks.

Flow problems can be generalized to multi-terminal or multi-commodity flows where multiple sources and sinks (the terminals) and several commodities are allowed. The simplification of such networks is not considered here, but it seems



(a) (b) (c) (d)

Fig. 2. All dashed edges are useless. (a) A tree. (b) A non-tree graph. (c) The graph in (b) with its biconnected components $A, B, \ldots, G$ and articulations points a, b, and c. (d) The block-cutpoint-tree for the graph in (b).

a natural extension of this work. Here, we concentrate on the problem of simplifying single-source, single-sink networks.

Obviously, it is impractical to eliminate useless edges by computing all possible s-to-t flows in G and identifying which edges get involved in none of all such flows. The following characterization, proven by Biedl, Brejová, and Vinař [4], is helpful to the development of a more efficient simplification algorithm.

Lemma 1 ([4]). *Given a flow network $G = (V, E, s, t)$, an edge $e \in E$ is useful if and only if there exists a simple path in G from s to t (called an (s, t)-path) that contains e.*

In this paper, we present what we believe is the first optimal algorithm for simplifying undirected flow networks. Our algorithm removes all useless edges from an undirected network in $O(|V|+|E|)$ time. We also improve the algorithm of Biedl, Brejová, and Vinař [4] for removing a special class of useless edges from a general directed network. We reduce their $O(|V|+|E|^2 \log |V|)$ time bound [4] by a factor of $\log |V|$, by replacing the main part of their algorithm by an algorithm for finding a dominance tree in a directed graph [3]. Both of our algorithms are actually quite simple.

The rest of the paper is organized as follows. Section 2 presents the algorithm for simplifying undirected networks. Section 3 gives our improvement over the algorithm in [4] for the directed network case. Section 4 concludes the paper.

2 Simplifying Undirected Networks

In this section, we present a linear time algorithm for simplifying an undirected flow network $G = (V, E, s, t)$. First, consider the graph in Figure 2(a). Since this graph is a tree, the task of removing all useless edges is straightforward: We simply find the unique simple path between s and t, and remove all edges that do not belong to this path. But in a general undirected network (e.g., see Figure 2(b)), such an (s, t)-path is not unique in the graph, so the solution becomes less obvious. Our idea is to reduce the problem on a general undirected graph to that on a tree. Without loss of generality (WLOG), we assume that G is connected.

Since every useful edge belongs to at least one simple (s,t)-path, all useless edges must belong to some "parts" of G such that each such part shares exactly one vertex with the set of all simple (s,t)-paths in G. Thus, to identify all useless edges, we consider a graph decomposition, namely, the decomposition of G into its *biconnected components*. A useful observation is that for any biconnected component in G, either all its edges are useful, and we call such a component a *useful biconnected component*, or all its edges are useless, and we call such a component a *useless biconnected component*. This observation enables us to reduce the problem of searching for useless edges to the problem of searching for useless biconnected components. We will model the latter problem by a *tree* $H(G)$, called the *block-cutpoint-tree* [10], defined on the biconnected components of G. The block-cutpoint-tree $H(G)$ represents the relationship among the biconnected components and articulations points in G, and helps make our simplification problem substantially easier.

2.1 Biconnected Components

Computing the biconnected components of an undirected graph is one of the most basic problems in graph theory [1, 7, 13].

We establish that for any biconnected component, all edges in the component are either useful or useless. This will allow us to search for useless biconnected components (i.e., sets of edges) instead of individual useless edges.

Lemma 2. *If an undirected flow network G is biconnected, then all edges in G are useful.*

Proof. Let e be any edge in G. We will show that there exists a simple path in G between s and t containing e. WLOG, suppose that G contains an edge f connecting s and t. If there is no such f, then we can add it to G, and G will remain biconnected. Since G is biconnected, it must contain a simple cycle that includes both e and f. Removing f from this cycle yields a simple path between s and t that includes e, which proves that e is useful. □

Lemma 3. *For any biconnected component B of an undirected flow network G, either all edges of B are useful or all of them are useless.*

Proof. If $B = G$, then by Lemma 2, B contains only useful edges. Assume $B \neq G$. We need to show that B cannot contain both useful and useless edges. Actually, we will show that if there exists one useful edge e in B, then all edges of B must be useful. Let e be a useful edge in B and let f be any other edge in B. Since e is useful, there exists a simple (s,t)-path P_e containing e. Let x and y be respectively the first and last vertices on P_e that belong to B. Since P_e is simple, we must have $x \neq y$. Let $P_{s,x}$ and $P_{y,t}$ be the simple sub-paths of P_e from s to x and from y to t, respectively. As in the proof of Lemma 2, we can use the edge (x,y). If G does not contain such an edge, then WLOG we can add (x,y) to B. In this way, we obtain a simple path P_f in B from x to y containing f. Concatenating $P_{s,x}$, P_f, and $P_{y,t}$ gives a simple (s,t)-path containing f, thus proving that f is useful. □

2.2 Block-Cutpoint-Tree

The *block-cutpoint-tree* of G represents the decomposition of G into its biconnected components. It was first introduced by Harary and Prins [10].

Definition 2. *The* block-cutpoint-tree *of G is a graph $H(G)$ that contains a vertex b_i for each biconnected component B_i and a vertex v_i for each articulation point a_i of G. An edge (v_i, b_i) is put in $H(G)$ if and only if a_i is an articulation point belonging to the biconnected component B_i.*

Harary and Prins [10] showed that if G is connected, then $H(G)$ is a tree; otherwise, it is a forest. Since we assume that G is connected, the graph $H(G)$ for G is a tree. Figure 2(b) shows an undirected graph G, Figure 2(c) the biconnected components of G, and Figure 2(d) the block-cutpoint-tree $H(G)$ of G. Let the nodes S and T in $H(G)$ represent the biconnected components of G containing s and t, respectively. Note that it is possible to have $S = T$. The unique simple path between S and T in $H(G)$ is denoted by P. Observe that every simple path between s and t in G is mapped to P in $H(G)$.

We will show below that the useless nodes in $H(G)$ correspond to the useless biconnected components of G.

Lemma 4. *A biconnected component B_i of G is useless if and only if the node b_i in $H(G)$ corresponding to B_i is useless in $H(G)$.*

Proof. First, assume b_i is useless in $H(G)$. If B_i is not useless in G, then it must contain an edge e which lies on some simple path between s and t in G. This path is mapped to the (unique) simple path P between S and T in $H(G)$. It thus follows that P contains b_i as a vertex, a contradiction with the assumption that b_i is useless in $H(G)$.

Suppose now that B_i is useless in G. If b_i is not useless in $H(G)$, then b_i must belong to P. But this implies that B_i contains at least one useful edge since a simple path between s and t in G must go through B_i. By Lemma 3, B_i must be useful, a contradiction. □

2.3 Our Algorithm

Based on the above lemmas, we can now simplify an undirected flow network G as follows.

1. Decompose G into biconnected components and construct the corresponding block-cutpoint-tree $H(G)$;
2. simplify $H(G)$ using a single depth-first search to identify the unique simple path P between S and T in $H(G)$; useful biconnected components of G are represented by the vertices of $H(G)$ lying on the simple path P;
3. remove all biconnected components of G that do not correspond to any vertices of $H(G)$ on the path P.

The correctness of the above algorithm follows from Lemma 4. It is easy to show that every step of the algorithm takes $O(|V| + |E|)$ time. Since there are only a constant number of steps, the total time for simplifying an undirected flow graph is $O(|V| + |E|)$.

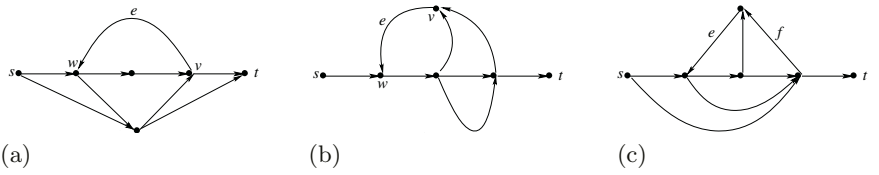


Fig. 3. (a) Edge e is useless but not s-or-t-useless. (b) Edge e is s-useless since all directed paths from s to v use w. (c) Removing the t-useless edge f creates a new s-useless edge e.

3 Simplifying General Directed Networks

In this section, we focus on dealing with the case of directed flow networks. Note that if a directed flow network $G = (V, E, s, t)$ is acyclic, then it is easy to remove all useless edges from G: Those are the edges that are not reachable from s and those that cannot reach t. All edges that both are reachable from s and can reach t are useful (since G is acyclic, such edges are all on some simple paths from s to t in G). Removing all useless edges from a directed acyclic flow network G can be easily done in $O(|V| + |E|)$ time, by performing depth-first search. Further, for a general directed flow network $G = (V, E, s, t)$ (which may contain cycles), it is straightforward to remove all (useless) edges that are not reachable from s and those that cannot reach t, in $O(|V| + |E|)$ time. In addition, determining whether a directed graph is acyclic can be done in linear time [1, 7, 13]. Hence, WLOG we assume that the given directed flow network G contains cycles, and every edge e of G lies on a certain (possibly non-simple) directed path from s to t.

Biedl, Brejová, and Vinař showed that the problem of simplifying a general directed flow network is NP-complete [4]. They also considered a modified problem: simplifying a directed flow network by removing only certain special useless edges called *s-useless* and *t-useless* edges [4]. According to their definition, an edge $e = (v, w)$ in G is *s-useless* if no path in G from s to w containing e is simple (e.g., see Figure 3(b)). Similarly, an edge $e = (v, w)$ is *t-useless* if no path in G from v to t containing e is simple. We say that e is *s-or-t-useless* if it is *s-useless*, *t-useless*, or both. Observe that there can be useless edges that are not s-or-t-useless (e.g., see Figure 3(a)). Such edges will not necessarily be removed in the modified problem.

It is sufficient for us to show the details on how to remove s-useless edges (removing t-useless edges is done in a similar way). The following characterization of s-useless edges serves as a basis for the algorithm of Biedl *et al.* [4].

Lemma 5 ([4]). *Let $G = (V, E, s, t)$ be a directed flow network each of whose edges belongs to some (possibly non-simple) path from s to t. An edge $e = (v, w)$ in G is s-useless if and only if all paths from s to v in G contain w. Equivalently, $e = (v, w)$ is s-useful if and only if there exists a simple (s, v)-path in G that does not contain w.*

The algorithm of Biedl *et al.* [4] proceeds iteratively, as follows: (1) remove all s-useless edges in G; (2) reverse the directions of all edges and remove all t-useless edges in G using the same method as for the s-useless edge case. However, removing s-useless edges may give rise to new t-useless edges, and *vice versa* (e.g., see Figure 3(c)). Thus, repeated iterations, each consisting of steps (1) and (2) above, are performed until all s-or-t-useless edges are eliminated. In the worst case, the number of repeated iterations can be $O(|E|)$. The algorithm in [4] makes use of several data structures such as interval trees [7] and dynamic trees [15], and performs three depth-first search traversals of the graph in each iteration. The time bound for each iteration is $O(|E| \log |V|)$. Therefore, the total time of the algorithm in [4] is $O(|V| + |E|^2 \log |V|)$.

We present a somewhat different and simpler method that takes only $O(|E|)$ time per iteration. Our approach also uses the characterization of s-useless edges in Lemma 5 and is based on the concept of *dominance* among vertices in directed graphs.

3.1 Dominance and s-Useless Edges

The concept of dominance among vertices in a directed graph has been studied quite extensively in the area of compilers [6]. The following definition and lemma are standard.

Definition 3. *Let $G = (V, E, s, t)$ be a directed flow network, and v and w be two distinct vertices in G. If every path from s to v in G passes through w, then we say that w* dominates *v.*

Lemma 6. *Let $G = (V, E, s, t)$ be a directed flow network. For any three distinct vertices v, w, and u in G, the following holds.*
1. If v dominates w and w dominates u, then v dominates u.
2. If vertices w and u both dominate v, then either u dominates w or w dominates u.

The above statements define a unique tree structure $D(G) = (V_D, E_D, s)$ rooted at s on G. This tree can be constructed in linear time using, for example, the algorithm of Alstrup *et al.* [3].

Definition 4. *Let $G = (V, E, s, t)$ be a directed flow network. The tree $D(G) = (V_D, E_D, s)$, called the* dominance tree *of G, is defined as follows: $V_D \subseteq V$, every edge $(v, w) \in E_D$ represents the relation of v immediately dominating w in G, and $D(G)$ is rooted at s.*

Using the dominance tree $D(G)$, we can determine which edges of G are s-useless. The following lemma is useful.

Lemma 7. *Let $G = (V, E, s, t)$ be a directed flow network. An edge $e = (v, w)$ in G is s-useless if and only if w dominates v.*

Proof. The proof follows directly from Definition 3 and Lemma 5. □

It turns out that we only need to check, using $D(G)$, the edges of G for the following relation: Whether the dominance relation holds for both end vertices of each edge in $D(G)$. The following lemma states this assertion.

Lemma 8. *Let $e = (v, w)$ be an edge of a directed flow network $G = (V, E, s, t)$. If v or w does not belong to the dominance tree $D(G)$ (i.e., $v \notin V_D$ or $w \notin V_D$), then e is s-useful.*

Proof. If v does not belong to $D(G)$, then v is not dominated by any other vertex in V. In particular, v is not dominated by w, and thus there is a simple path from s to v not containing w. Therefore, $e = (v, w)$ is s-useful. On the other hand, if w does not belong to $D(G)$, then w does not dominate any other vertex in G. In particular, w does not dominate v, and again there is a simple path from s to v not containing w, proving that $e = (v, w)$ is s-useful. □

Lemma 8 shows that a necessary condition for an edge $e = (v, w)$ to be s-useless is that both of its end vertices belong to $D(G)$. However, edges with both end vertices in $D(G)$ need not be s-useless. The next lemma specifies the sufficient condition.

Lemma 9. *Let $G = (V, E, s, t)$ be a directed flow network, and w and v be two distinct vertices in G. Then w dominates v if and only if both v and w belong to V_D and w is a proper ancestor of v in $D(G)$.*

Proof. The proof follows immediately from Lemma 6. □

Thus, to determine whether an edge $e = (v, w)$ with both of its end vertices in $D(G)$ is s-useless, it suffices to check whether w is a proper ancestor of v in the tree $D(G)$. In the worst case, we may need to check for $O(|E|)$ edges in each iteration of our algorithm. In order for the time bound of every iteration to be $O(|E|)$ time, the checking on each edge must be done in $O(1)$ time. These checkings can be carried out by using the *lowest common ancestor queries* on $D(G)$ [11, 14].

The next definition is included for completeness.

Definition 5. *Let $T = (V_T, E_T, s)$ be a tree rooted at s, and v_1 and v_2 be two arbitrary vertices in V_T. A vertex w is called the* lowest common ancestor *of v_1 and v_2 if w is an ancestor of both v_1 and v_2 in T and no proper descendant of w is an ancestor of both v_1 and v_2 in T.*

The lowest common ancestor of vertices v_1 and v_2 is denoted by $LCA(v_1, v_2)$. Note that if v_1 is an ancestor of v_2 in T, then $LCA(v_1, v_2) = v_1$. It is well known that any rooted tree T can be preprocessed in linear time to build a data structure, which enables each lowest common ancestor query on T to be performed in $O(1)$ time [11, 14].

3.2 Our Algorithm

Our algorithm consists of a series of iterations, each removing all s-useless and t-useless edges found in the version of the graph for the corresponding iteration. The elimination of t-useless edges in each iteration is done by the same procedure as for s-useless edges, except that it is performed on a graph in which the directions of all edges and the roles of the two vertices s and t are reversed. Note that removing s-useless edges in the graph may create new t-useless edges, and *vice versa* [4]. The iterative process continues as long as new s-or-t-useless edges are generated in the graph by the previous iteration.

Each iteration of the algorithm performs the following steps for removing s-useless edges.

1. Construct the dominance tree $D(G')$ for the current version G' of the graph (initially, $G' = G$), and the LCA data structure for $D(G')$;
2. for each edge (v, w) in G' with both end vertices in $D(G')$, perform the query $LCA(v, w)$ to determine whether the vertex w dominates the vertex v in G' (for an edge (v, w) in G', if $LCA(v, w) = w$ in $D(G')$, then w dominates v in G' and (v, w) is s-useless; in all other cases, (v, w) is s-useful in the current graph G');
3. remove from G' all edges determined to be s-useless in Step 2.

The algorithm terminates if in Step 2 of the above iteration procedure, no s-or-t-useless edges are found.

The correctness of the algorithm follows from Lemmas 7, 8, and 9. In each iteration, the construction of the dominance tree $D(G')$ takes $O(|E|)$ time. The construction of the LCA data structure for $D(G')$ takes $O(|V|)$ time. Checking which edges are s-or-t-useful takes $O(|E|)$ time ($O(1)$ time per edge, with one LCA query for every edge in G'). Hence, each iteration runs in $O(|V|+|E|)$ time. Since as many as $O(|E|)$ iterations may be needed to remove all s-or-t-useless edges [4], the total time of our algorithm is $O(|E| * (|V| + |E|))$.

4 Conclusion

We have presented two algorithms for simplifying flow networks. These algorithms remove edges that cannot contribute to any source-to-sink flow in the given network. Since such simplification may lead to a considerable reduction in the computation time of flows, our results are useful in the applied settings of flow algorithms.

While the case on undirected networks has been solved optimally, the case on directed networks may be further improved. The simplification of planar directed graphs is polynomial time solvable, which was shown by Biedl *et al.* [4] using an $O(|V|^2)$ time algorithm. A motivation for considering the planar directed graph case was due to a result of Weihe [16], in which he proposed an $O(|V| \log |V|)$ time maximum flow algorithm for planar directed graphs. Weihe's algorithm requires that the input graph contains no useless edges. So far, however, no $O(|V| \log |V|)$

time algorithm for removing all useless edges in planar directed graphs is known [5]. The quadratic time bound of Biedl *et al.*'s algorithm [4] does not satisfy the need of Weihe's maximum flow algorithm [16]. It is thus desirable to design an $O(|V|\log|V|)$ time algorithm for simplifying planar directed networks. If all useless edges in planar directed networks can be removed in $O(|V|\log|V|)$ time, then Weihe's algorithm will be the first $O(|V|\log|V|)$ time maximum flow solution for planar directed graphs.

Developing a provably good polynomial time approximation algorithm for simplifying general directed flow networks is also an interesting problem.

References

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.
2. R. Ahuja, T. Magnanti, J. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, 1993.
3. S. Alstrup, D. Harel, P. Lauridsen, and M. Thorup, Dominators in linear time, *SIAM Journal on Computing*, 28(6):2117-2132, 1999.
4. T. Biedl, B. Brejová, and T. Vinař, Simplifying flow networks, *Proc. 25th Intl. Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol. 1893, pp. 202-211, 2000.
5. B. Brejová and T. Vinař, Weihe's algorithm for maximum flow in planar graphs (project report), University of Waterloo, Course CS760K, 1999.
6. K. Cooper, T. Harvey, and K. Kennedy, A simple, fast dominance algorithm, *Softw. Pract. Exper.*, 4:1-10, 2001.
7. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, McGraw-Hill, 2nd ed., 2001.
8. L.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.
9. A. Goldberg, Recent developments in maximum flow algorithms, Technical Report #98-045, NEC Research Institute, 1998.
10. F. Harary and G. Prins, The block-cutpoint-tree of a graph, *Publ. Math. Debrecen*, 13:103-107, 1966.
11. D. Harel and R.E. Tarjan, Fast algorithms for finding nearest common ancestors, *SIAM J. Comput.*, 13:338-355, 1984.
12. E.L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, 1976.
13. U. Manber, *Introduction to Algorithms, A Creative Approach*, Addison-Wesley, 1989.
14. B. Schieber and U. Vishkin, On finding lowest common ancestors: Simplification and parallelization, *SIAM Journal on Computing*, 17(6):1253–1262, 1988.
15. D.D. Sleator and R.E. Tarjan, A data structure for dynamic trees, *Journal of Computer and System Sciences*, 26(3):362-391, 1983.
16. K. Weihe, Maximum (s,t)-flows in planar networks in $O(|V|\log|V|)$ time, *Journal of Computer and System Sciences*, 55(3):454-475, 1997.

Approximation Algorithms for the *b*-Edge Dominating Set Problem and Its Related Problems

Takuro Fukunaga and Hiroshi Nagamochi

Department of Applied Mathematics and Physics
Graduate School of Informatics, Kyoto University
{takuro,nag}@amp.i.kyoto-u.ac.jp

Abstract. The edge dominating set problem is one of the fundamental covering problems in the field of combinatorial optimization. In this paper, we consider the b-edge dominating set problem, a generalized version of the edge dominating set problem. In this version, we are given a simple undirected graph $G = (V, E)$ and a demand vector $b \in \mathbb{Z}_+^E$. A set F of edges in G is called a b-edge dominating set if each edge $e \in E$ is adjacent to at least $b(e)$ edges in F, where we allow F to contain multiple copies of edges in E. Given a cost vector $w \in \mathbb{Q}_+^E$, the problem asks to find a minimum cost of a b-edge dominating set. We first show that there is a $\frac{8}{3}$-approximation algorithm for this problem. We then consider approximation algorithms for other related problems.

1 Introduction

Let $G = (V, E)$ be a simple undirected graph. Moreover, let $\mathbb{Z}_+$, $\mathbb{Q}_+$ and $\mathbb{R}_+$ denote the sets of nonnegative integers, rational numbers and real numbers, respectively. We say that an edge $e = (u, v)$ *dominates* edges incident to u and v, and define an *edge dominating set* (EDS) to be a set F of edges such that each edge in E is dominated by at least one edge in F. Given a cost vector $w \in \mathbb{Q}_+^E$ together with G, the *EDS problem* asks to find an EDS with the minimum cost. The problem with a cost vector w with $w(e) = 1$, $e \in E$ is called the cardinality case; otherwise the problem is called the cost case. The EDS problem is one of the fundamental covering problems in the field of combinatorial optimization and has some useful applications [1, 11].

It is known that the cardinality case of the EDS problem is NP-hard even for some restricted classes of graphs such as planar or bipartite graphs of maximum degree 3 [11]. For the cardinality case of the EDS, an arbitrary algorithm that outputs a maximal matching is a 2-approximation algorithm [2, 6].

Carr et al. [2] presented a 2.1-approximation algorithm for the cost case of the EDS problem. This algorithm first constructs an instance of the minimum cost *edge cover problem* from the original instance and then finds an optimal edge cover in the resulting instance. A key property for this method is that an edge cover in the resulting instance is also an EDS for the original instance and

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 747–756, 2005.

that its cost is at most 2.1 times of the minimum cost of an EDS in the original instance. The property is proved based on a relation between the fractional EDS polyhedron and the edge cover polyhedron. The former is a polyhedron containing all incidence vectors of EDSs, which may not be the convex hull of these vectors. In contrast the edge cover polyhedron is the convex hull of all incidence vectors of edge covers, which is shown to be an integer polyhedron [10]. Afterward by using a refined EDS polyhedron, Fujito and Nagamochi [4] gave a 2-approximation algorithm to the cost case of the EDS problem. Moreover, Könemann et al. proposed 3-approximation algorithms for two related problems; they ask to find a minimum cost EDS which forms a tree/tours [7]. Note that, in the above algorithms, a linear program relaxation of the integer program formulation is used, but an output solution is not constructed directly from solutions of the linear program or its dual problem.

In this paper, we consider the following three natural extensions of the EDS or the edge cover, which will be expected to allow more flexible modelings in practice.

- The *b-edge dominating set* (b-EDS) problem, a capacitated version of the EDS. Given a graph $G = (V, E)$, a demand vector $b \in \mathbb{Z}_+^E$ and a cost vector $w \in \mathbb{Q}_+^E$, the problem asks to find a minimum cost b-EDS, where a set F of edges in G is called a b-EDS if each $e \in E$ is adjacent to at least $b(e)$ edges in F (we allow F to contain multiple copies of edges in E).
- The *EDS problem in hypergraphs* (HEDS problem), an extension of the EDS. Given a hypergraph $H = (V, E)$ and a cost vector $w \in \mathbb{Q}_+^E$, the problem asks to find a minimum cost hyperedge set F such that each hyperedge $e \in E$ is either contained in F or adjacent to a hyperedge in F.
- The *b-edge cover with degree constraints over subsets*, an extension of the edge cover. Given a graph $G = (V, E)$, a cost vector $w \in \mathbb{Q}_+^E$, a family $\mathcal{S} \subseteq 2^V$ of vertex sets, and a capacity vector $b \in \mathbb{Z}_+^{\mathcal{S}}$, the problem asks to find a minimum cost edge set F such that the sum of degrees in graph (V, F) over $S \in \mathcal{S}$ is at least $b(S)$, where F can contain multiple copies of edges.

In this paper, we present approximation algorithms for the above problems by investigating polyhedral structures of the convex hulls of their feasible solutions. As a result, we show that the b-EDS is approximable within a factor of $2\left(1 + \frac{1}{2\lfloor 3\beta/2 \rfloor + 1}\right)$ $(\leq \frac{8}{3})$, where $\beta = \min_{e \in E, b(e) \neq 0} b(e)$ while the HEDS and the b-edge cover with degree constraints over subsets are approximable within factors of $k\theta_k$, and $\frac{4}{3}h$, respectively, where k is the maximum size of hyperedges in H, θ_k denotes the k-th harmonic number and h is the maximum cardinality of $S \in \mathcal{S}$.

The paper is organized as follows. Section 2 introduces some notations. Sections 3, 4 and 5, respectively, describe formulations of the b-EDS problem, the HEDS problem and the b-edge cover with degree constraints over subsets and present approximation algorithms for these problems by investigating their polyhedral structures.

2 Preliminaries

We denote by $\theta_k \in \mathbb{Q}_+$ the k-th harmonic number $\sum_{i=1}^{k} \frac{1}{i}$. Let $G = (V, E)$ denote a simple undirected graph with a vertex set V and an edge set E. An edge $e = (u, v) \in E$ in G is defined as a pair of distinct vertices u and v. Let $H = (V, E)$ denote a hypergraph, where an edge is defined by a set of two or more vertices and an edge in H may be called a hyperedge. For a vertex v, $\delta(v)$ denotes the set of edges incident to v. For an edge e, $\delta(e)$ denotes the set of edges incident to vertices contained in e, i.e., $\delta(e) = \{e' \in E |\ e \cap e' \neq \phi\}$. For a subset $S \subseteq V$, $\delta(S)$ denotes the set of edges $e = (u, v)$ with $u \in S$ and $v \in V - S$, and $E[S]$ denotes the set of edges contained in S, i.e., $E[S] = \{e \in E |\ e \subseteq S\}$. Let x be an $|E|$-dimensional nonnegative real vector, i.e., $x \in \mathbb{R}_+^E$. We indicate the entry in x corresponding to an edge e by $x(e)$. For a subset F of E, we denote $x(F) = \sum_{e \in F} x(e)$. For an edge set F such that each edge $e' \in F$ corresponds to an edge $e \in E$, $x\langle F \rangle \in \mathbb{R}_+^F$ denotes a projection of x to F, i.e., $x\langle F \rangle (e') = x(e)$ for all $e' \in F$.

3 b-EDS Problem

3.1 b-Edge Cover Problem and b-Edge Cover Polyhedron

For a graph $G = (V, E)$, a demand vector $b \in \mathbb{Z}_+^E$, and a cost vector $w \in \mathbb{Q}_+^E$, an integer program of the b-EDS is given as

$$\begin{aligned} &\text{minimize} && w^T x \\ &\text{subject to} && x(\delta(e)) \geq b(e) \text{ for each } e \in E, \\ & && x \in \mathbb{Z}_+^E. \end{aligned} \tag{1}$$

A vector $x \in \mathbb{Z}_+^E$ satisfying (2) is called a b-EDS.

We now review some results on the *b-edge cover problem*, which is another important covering problem. This problem consists of a simple undirected graph $G = (V, E)$, a demand vector $b \in \mathbb{Z}_+^V$ defined on V and a cost vector $w \in \mathbb{Q}_+^E$. An integer vector $x \in \mathbb{Z}_+^E$ is called a *b-edge cover* if $x(\delta(v)) \geq b(v)$ for each $v \in V$. The objective of the b-edge cover problem is to find a minimum cost b-edge cover, which is formulated as

$$\begin{aligned} &\text{minimize} && w^T x \\ &\text{subject to} && x(\delta(v)) \geq b(v) \text{ for each } v \in V, \\ & && x \in \mathbb{Z}_+^E. \end{aligned} \tag{2}$$

There exists a polynomial time algorithm for this problem [9]. Furthermore, we know that this problem has an equivalent linear program formulation, where convex hull of all feasible solutions is characterized by the following set of inequalities:

$$\begin{aligned} &\text{(a) } x(e) \geq 0 && \text{for each } e \in E, \\ &\text{(b) } x(\delta(v)) \geq b(v) && \text{for each } v \in V, \\ &\text{(c) } x(E[U]) + x(\delta(U)) \geq \left\lceil \tfrac{b(U)}{2} \right\rceil && \text{for each } U \subseteq V \text{ with odd } b(U). \end{aligned}$$

Let $\mathrm{EC}(G, b)$ denote the polyhedron represented by these inequalities. It is proven that $\mathrm{EC}(G, b)$ is an integer polyhedron [10], i.e., its extreme points are all integer vectors.

3.2 Approximation Algorithm

To approximate the b-EDS problem, we use its LP relaxation. The set $\mathrm{EDS}(G, b)$ of all feasible solutions of the relaxed problem is defined by the set of vectors $x \in \mathbb{R}_+^E$ such that

$$\begin{array}{ll} \text{(d) } x(e) \geq 0 & \text{for each } e \in E, \\ \text{(e) } x(\delta(e)) \geq b(e) & \text{for each } e \in E. \end{array}$$

In general, $\mathrm{EDS}(G, b)$ is not an integer polyhedron, implying that any minimum cost vector x^* in $\mathrm{EDS}(G, b)$ may not be integer solutions. The cost of an optimal solution in $\mathrm{EDS}(G, b)$ is a lower bound on the minimum cost of a given instance (G, b, w). Given an instance (G, b, w) of the b-EDS problem, we first construct constructs an instance of the b-edge cover and then computes an optimal solution as an approximate solution to the input instance. The algorithm is described as follows.

Algorithm DOMINATE

Input: A simple undirected graph $G = (V, E)$, a demand vector $b \in \mathbb{Z}_+^E$, and a cost vector $w \in \mathbb{Q}_+^E$.

Output: A b-EDS to instance (G, b, w).

Step 1: Compute an optimal solution $x^* \in \mathbb{R}_+^E$ to the linear program that minimizes $\min w^T x$ subject to $x \in \mathrm{EDS}(G, b)$.

Step 2: For each edge $e = (u, v) \in E$, let $b'_{x^*}(u, e) := b(e)$ and $b'_{x^*}(v, e) := 0$ if $x^*(\delta(u)) \geq x^*(\delta(v))$, and let $b'_{x^*}(u, e) := 0$ and $b'_{x^*}(v, e) := b(e)$ otherwise.

Step 3: For each vertex $v \in V$, let $\tilde{b}_{x^*}(v) := \max_{e \in \delta(v)} b'_{x^*}(v, e)$.

Step 4: Compute a minimum cost $\tilde{b}_{x^*}$-edge cover $\bar{x} \in \mathbb{Z}_+^E$ for G and w, and output $\bar{x}$ as a b-EDS to (G, b, w).

Note that $\bar{x} \in \mathrm{EC}(G, \tilde{b}_{x^*})$, i.e., $\bar{x}$ is a $\tilde{b}_{x^*}$-edge cover in G. We first show that $\bar{x}$ is a b-EDS. For each $e = (u, v) \in E$, $\tilde{b}_{x^*}(u) \geq b(e)$ or $\tilde{b}_{x^*}(v) \geq b(e)$ holds. Then

$$\bar{x}(\delta(e)) \geq \max\{\bar{x}(\delta(u)), \bar{x}(\delta(v))\} \geq \max\{\tilde{b}_{x^*}(u), \tilde{b}_{x^*}(v)\} \geq b(e)$$

holds. Hence, $\bar{x}$ is a b-EDS and algorithm DOMINATE outputs a feasible solution. We then analyze the approximation factor of algorithm DOMINATE by establishing a relation between $\mathrm{EDS}(G, b)$ and $\mathrm{EC}(G, \tilde{b}_{x^*})$. In the following discussion, we suppose that $b(e) \geq 1$ for at least one edge $e \in E$ because, if $b(e) = 0$ for all edges $e \in E$, DOMINATE apparently outputs the optimal solution $\bar{x} = 0^E$.

Lemma 1. *Let x be a vector in* $\mathrm{EDS}(G, b)$*, and $\tilde{b}_x \in \mathbb{Z}_+^V$ be a vector constructed from x by Step 3 of algorithm* DOMINATE*. Then vector $2x \in \mathbb{R}_+^E$ satisfies conditions* (a) *and* (b) *for* $\mathrm{EC}(G, \tilde{b}_x)$*.*

Proof. Let $x \in \mathrm{EDS}(G,b)$. Then vector $2x$ satisfies condition (a) for $\mathrm{EC}(G,\tilde{b}_x)$ because $x \in \mathbb{R}_+^E$ holds by (d) for $\mathrm{EDS}(G,b)$. We now show that $2x$ satisfies (b), i.e., $2x(\delta(v)) \geq \tilde{b}_x(v)$ for all $v \in V$. Let v be a vertex in V. Then there is an edge $e = (u,v) \in E$ such that $\tilde{b}_x(v) = b'_x(v,e)$. If $b'_x(v,e) = 0$, then we have $2x(\delta(v)) \geq 0 = \tilde{b}_x(v)$ since $x \in \mathbb{R}_+^E$ holds. Therefore, let us assume $b'_x(v,e) > 0$. Then $b'_x(v,e) = b(e)$ and $x(\delta(v)) \geq x(\delta(u))$ hold. Now $x(\delta(e)) \geq b(e)$ holds by (e) for $\mathrm{EDS}(G,b)$, which implies $x(\delta(v)) + x(\delta(u)) \geq b(e) + x(e)$ holds. Then we have

$$2x(\delta(v)) \geq x(\delta(u)) + x(\delta(v)) \geq b(e) + x(e) \geq b(e) = b'_x(v,e) = \tilde{b}_x(v).$$

Therefore, (b) also holds for $2x$. □

Lemma 2. *For a simple undirected graph $G = (V,E)$ and a demand vector $b \in \mathbb{Z}_+^V$, let $\beta = \min_{v \in V, b(v) \neq 0} b(v)$. Then, for any vector $x' \in \mathbb{R}_+^E$ satisfying conditions* (a) *and* (b) *for* $\mathrm{EC}(G,b)$*, vector*

$$y = \left(1 + \frac{1}{2\lfloor 3\beta/2 \rfloor + 1}\right) x' \in \mathbb{R}_+^E$$

satisfies condition (c) *for* $\mathrm{EC}(G,b)$.

Proof. Let U be a subset of V such that $b(U)$ is odd. It suffices to show that

$$y(E[U]) + y(\delta(U)) \geq \left\lceil \frac{b(U)}{2} \right\rceil \tag{3}$$

holds. If U contains a vertex v such that $b(v) = 0$, then (3) follows inductively from that $y(E[U']) + y(\delta(U')) \geq \left\lceil \frac{b(U')}{2} \right\rceil$ for $U' = U \backslash \{v\}$ since $y(E[U]) + y(\delta(U)) \geq y(E[U']) + y(\delta(U'))$ and $b(U) = b(U')$ hold. Hence we assume without loss of generality that $b(v) \geq \beta$ for all $v \in U$. Moreover, if $|U| = 1$, then (c) is implied by (b) since for $U = \{v\}$, $y(E[U]) + y(\delta(U)) = y(\delta(v)) \geq x'(\delta(v)) \geq b(v) \geq \left\lceil \frac{b(v)}{2} \right\rceil$. We now consider the case of $|U| = 2$. Let $U = \{v_1, v_2\}$. Since $b(U) = b(v_1) + b(v_2)$ is odd, $b(v_1) \neq b(v_2)$ holds, where we assume without loss of generality $b(v_1) > b(v_2)$. Then

$$\left\lceil \frac{b(U)}{2} \right\rceil = \left\lceil \frac{b(v_1) + b(v_2)}{2} \right\rceil \leq b(v_1).$$

We have

$$x'(E[U]) + x'(\delta(U)) \geq x'(\delta(v_1))$$

because $E[U] \cup \delta(U) \supseteq \delta(v_1)$. Since x' satisfies $x'(\delta(v_1)) \geq b(v_1)$ by (b), we have

$$\begin{aligned} y(E[U]) + y(\delta(U)) &\geq x'(E[U]) + x'(\delta(U)) \\ &\geq x'(\delta(v_1)) \geq b(v_1) \geq \left\lceil \frac{b(v_1) + b(v_2)}{2} \right\rceil = \left\lceil \frac{b(U)}{2} \right\rceil. \end{aligned}$$

In what follows, we assume that $|U| \geq 3$ and $b(v) \geq \beta$ for all $v \in U$.

Since $x'(\delta(v)) \geq b(v)$ holds for all $v \in U$ by (b) for $\mathrm{EC}(G, b)$, we have

$$2x'(E[U]) + x'(\delta(U)) = \sum_{v \in U} x'(\delta(v)) \geq b(U),$$

for which it holds

$$x'(E[U]) + x'(\delta(U)) \geq \frac{b(U) + x'(\delta(U))}{2} \geq \frac{b(U)}{2}.$$

To show (3), we only have to prove that

$$\frac{\lceil b(U)/2 \rceil}{b(U)/2} = 1 + \frac{1}{b(U)} \leq 1 + \frac{1}{2\lfloor 3\beta/2 \rfloor + 1},$$

or equivalently

$$b(U) \geq 2\lfloor 3\beta/2 \rfloor + 1. \tag{4}$$

From the assumption, $b(U) \geq 3\beta$ holds. Moreover, since $b(U)$ is odd, $b(U) \geq 3\beta + 1$ if 3β is even. This implies (4). □

Theorem 1. *Let* $\beta = \min_{e \in E, b(e) \neq 0} b(e)$. *Algorithm* DOMINATE *delivers an approximate solution of a cost within a factor of*

$$\rho = 2\left(1 + \frac{1}{2\lfloor 3\beta/2 \rfloor + 1}\right)\left(\leq \frac{8}{3}\right)$$

to the b-EDS problem.

Proof. Let $\bar{x} \in \mathbb{Z}_+^E$ be a vector obtained by algorithm DOMINATE. We have already observed that $\bar{x}$ is a b-EDS to (G, b, w). We show that $\bar{x}$ is a ρ-approximate solution. We denote by OPT the minimum cost of a b-EDS for (G, b, w). Let $x^* \in \mathbb{R}_+^E$ be a vector computed in Step 1 of DOMINATE. Since $\mathrm{EDS}(G, b)$ contains a minimum cost b-EDS, which minimizes $w^T x^*$ over $\mathrm{EDS}(G, b)$, it holds $w^T x^* \leq$ OPT. By Lemma 1, vector $2x^*$ satisfies conditions (a) and (b) for $\mathrm{EC}(G, \tilde{b}_{x^*})$. Since $b(e) \geq \beta$ for all $e \in E$ such that $b(e) \neq 0$, we see that $\tilde{b}_{x^*}(v) \geq \beta$ or $\tilde{b}_{x^*}(v) = 0$ holds for each $v \in V$. Therefore, from Lemma 2, we have $\rho x \in \mathrm{EC}(G, \tilde{b}_{x^*})$. Since algorithm DOMINATE outputs a solution $\bar{x}$ of the minimum cost over all vectors in $\mathrm{EC}(G, \tilde{b}_{x^*})$, we have $w^T \bar{x} \leq \rho w^T x^*$, from which it follows $w^T \bar{x} \leq \rho$OPT, as required. □

We have an instance that indicates that the analysis of Theorem 1 is tight in the case of $\beta = 1$ (the detail is omitted due to the space limitation). Moreover, algorithm DOMINATE achieves a better approximation factor when the edge set E contains no edge e with $b(e) = 0$ as follows.

Theorem 2. *For a demand vector* $b \in \mathbb{Z}_+^E$ *such that* $\beta = \min_{e \in E} b(e) \geq 1$, *algorithm* DOMINATE *delivers an approximate solution of a cost within a factor of*

$$\rho = 2\left(1 + \frac{1}{4\beta + 1}\right)\left(\leq \frac{12}{5}\right)$$

to the b-EDS problem.

Proof. Omitted due to the space limitation. □

In the rest of this section, we investigate the approximation guarantee of DOMINATE in some more restricted cases. We first consider the case of bipartite graphs.

Theorem 3. DOMINATE *is a* 2*-approximation algorithm for the b-EDS problem in bipartite graphs.*

Proof. For bipartite graphs, the edge cover polytopes are determined by only inequalities (a) and (b) [10]. Hence the theorem follows from Lemma 1. □

When b takes the same value for all edges, we show that a better guarantee can be derived as follows.

Lemma 3. *Let* $x \in \mathbb{R}_+^E$ *be a vector in* $\mathrm{EDS}(G, b)$*. If* $b(e) = \beta \geq 1$ *for all* $e \in E$*, then* ρx *belongs to* $\mathrm{EC}(G, \tilde{b}_x)$*, where* $\rho = 2.1$ *for* $\beta = 1$ *and* $\rho = 2$ *for* $\beta \geq 2$*.*

Proof. Omitted due to the space limitation. □

Lemma 3 directly implies the following theorem.

Theorem 4. *Suppose that* $b(e) = \beta$ *for all* $e \in E$*. Then algorithm* DOMINATE *delivers an approximate solution of a cost within a factor of* 2.1 *if* $\beta = 1$ *or a factor of* 2 *if* $\beta \geq 2$*.*

4 Hyperedge Dominating Set Problem

Given a hypergraph $H = (V, E)$ and a cost vector $w \in \mathbb{Q}_+^E$, the problem is formulated as

$$\begin{aligned} &\text{minimize} && w^T x \\ &\text{subject to} && x(\delta(e)) \geq 1 \text{ for each } e \in E, \\ &&& x \in \mathbb{Z}_+^E. \end{aligned} \tag{5}$$

An HEDS is defined as a vector $x \in \mathbb{Z}_+^E$ such that $x(\delta(e)) \geq 1$ for each $e \in E$. In addition, we call the LP relaxation of the HEDS problem *the fractional HEDS problem* and its feasible solution a *fractional HEDS*.

To obtain an approximate solution to the HEDS problem, we transform an instance of the HEDS problem to an instance of the *set cover problem*. The set cover problem is considered as a hypergraph version of the edge cover problem. A hyperedge set $F \subseteq E$ is called a *set cover* of hypergraph $H = (V, E)$ if $\cup_{e \in F} e = V$ and the set cover problem asks to find a minimum cost set cover. Given a hypergraph $H = (V, E)$ and a cost vector $w \in \mathbb{Q}_+^E$, the formulation of the set cover problem is given as follows.

$$\begin{aligned} &\text{minimize} && w^T x \\ &\text{subject to} && x(\delta(v)) \geq 1 \text{ for each } v \in V, \\ &&& x \in \mathbb{Z}_+^E. \end{aligned} \tag{6}$$

Note that this problem is proven to be NP-hard [5]. Moreover, the LP relaxation of the set cover problem is called the fractional set cover problem and its feasible solution is called a fractional set cover. It is known that a simple greedy algorithm finds an approximate solution for the set cover problem, and that the cost of the solution is bounded in terms of the minimum cost of a fractional set cover, as described in the following theorem.

Theorem 5. [3, 8] *Let $w \in \mathbb{Q}_+^E$ be a given cost vector, $\hat{x}$ be a minimum cost fractional set cover for a hypergraph $H = (V, E)$, and k be the maximum size of a hyperedge in H. Then a set cover whose cost is at most $\theta_k w^T \hat{x}$ can be obtained in polynomial time.*

Since the HEDS problem is a subclass of the set cover problem, the HEDS problem can be reduced to the set cover problem directly. Let $(H = (V, E), w)$ be a given instance of the HEDS problem. Construct a hypergraph $H' = (V', E')$ such that its vertex set V' consists of vertices v'_e corresponding to its edges $e \in E$ and edge set E' consists of $e'_e = \{v'_{e''} | e'' \in \delta(e)\}$ corresponding to $\delta(e)$. A component of the cost vector $w'(e'_e)$ is set to be $w(e)$. Then, it is easy to see that a set cover for (H', w') gives an HEDS for (H, w) of same cost and vice versa. Let d be the maximum size of a hyperedge in E', i.e., the maximum size of $\delta(e)$ for each $e \in E$, where $d = O(|V|^k)$ holds for the maximum size k of a hyperedge in H. By Theorem 5, this direct reduction gives a θ_d-approximation algorithm for the HEDS problem. Note that $\theta_d = O(k \log |V|)$.

In our algorithm, we transform an instance of the HEDS problem into an instance of the set cover problem. The algorithm is described as follows.

Algorithm HYPER
Input: A hypergraph $H = (V, E)$ and a cost vector $w \in \mathbb{Q}_+^E$.
Output: An HEDS for H.
Step 1: Find a minimum cost solution $x^* \in \mathbb{R}^E$ to the fractional HEDS problem for (H, w).
Step 2: Let $V' := \{v \in V \mid x^*(\delta(v)) = \max_{u \in e} x^*(\delta(u)) \text{ for some } e \in E\}$, $E' := \{e \cap V' \mid e \in E\}$, and $w' := w\langle E' \rangle \in \mathbb{Q}_+^{E'}$.
Step 3: Find a set cover $\bar{x}$ for hypergraph $H_{x^*} = (V', E')$ such that $w'^T \bar{x}$ is at most θ_k times the minimum cost of a fractional set cover, and output $\bar{x}\langle E \rangle$ as an HEDS for H.

To prove that the approximation factor of algorithm HYPER is $k\theta_k$ by using Theorem 5, we show that for a vector x^* obtained in Step 1, vector kx^* is a fractional set cover to (H_{x^*}, w').

Lemma 4. *Let $x \in \mathbb{R}^E$ be a fractional* HEDS *for a hypergraph $H = (V, E)$, $H_x = (V', E')$ be a hypergraph obtained in Step 3 of* HYPER *from x and k be the maximum hyperedge size of H. Then vector $kx\langle E' \rangle \in \mathbb{R}^{E'}$ is a fractional set cover for H_x.*

Proof. Suppose that $v \in V'$ is a vertex in a hyperedge $e \in E'$ such that $x(\delta(v)) \geq x(\delta(u))$ for each $u \in e$. Since $\sum_{u \in e} x(\delta(u)) \geq x(\delta(e)) \geq 1$, we have $x(\delta(v)) \geq 1/k$.

Therefore $kx\langle E'\rangle(\delta(v)) \geq 1$, which means that $kx\langle E'\rangle \in \mathbb{R}^{E'}$ is a fractional set cover for H_x. □

Theorem 6. *The algorithm* HYPER *achieves approximation factor of* $k\theta_k$ *for the* HEDS *problem, where* k *is the maximum size of a hyperedge.*

Proof. Omitted due to the space limitation. □

Note that the approximation factor $k\theta_k = O(k \log k)$ of algorithm HYPER is superior to that of the algorithm obtained from the direct reduction if $k\theta_k < \theta_d$, i.e., H is a dense hypergraph such that $d = \Omega(|V|^k)$.

5 *b*-Edge Cover with Degree Constraints over Subsets

Given an undirected graph $G = (V, E)$, a cost vector $w \in \mathbb{Q}_+^E$, a family $\mathcal{S} \subseteq 2^V$ of subsets of V, and a demand vector $b \in \mathbb{Z}_+^{\mathcal{S}}$, the b-edge cover with degree constraints over subsets is formulated by

$$\begin{array}{ll} \text{minimize} & w^T x \\ \text{subject to} & \sum_{v \in S} x(\delta(v)) \geq b(S) \text{ for each } S \in \mathcal{S}, \\ & x \in \mathbb{Z}_+^E. \end{array} \tag{7}$$

Note that if $\mathcal{S} = \{\{v\} | v \in V\}$, then problem (7) is equivalent to the b-edge cover problem (2). If $\mathcal{S} = \{\{u, v\} | (u, v) \in E\}$, then problem (7) seems similar to the b-EDS problem, but its constraint $x(\delta(u)) + x(\delta(v)) \geq b(e)$ on each $e = (u, v) \in E$ is different from the constraint $x(\delta(e)) \geq b(e)$ for the b-EDS. Let DC(G,b) denote the set of all vectors $x \in \mathbb{R}_+^E$ satisfying inequality in (7), i.e., the relaxation of the covering problem. We show that problem (7) is approximable by the next algorithm.

Algorithm COVER

Input: A simple undirected graph $G = (V, E)$, a cost vector $w \in \mathbb{Q}_+^E$, a family $\mathcal{S} \subseteq 2^V$ of subsets of V, and a demand vector $b \in \mathbb{Z}_+^{\mathcal{S}}$.

Output: A vector $x \in \mathbb{Z}_+^E$ feasible to the covering problem (7).

Step 1: Find a minimum cost vector $x^* \in \mathrm{DC}(G, b)$.

Step 2: For each $S \in \mathcal{S}$, $b'_{x^*}(v, S) := b(S)$ if $x^*(\delta(v)) \geq x^*(\delta(u))$ for all $u \in S$ and $b'_{x^*}(v, S) := 0$ otherwise.

Step 3: For each $v \in V$, $\tilde{b}_{x^*}(v) := \max_{S \in \mathcal{S}: v \in S} b'(v, S)$.

Step 4: Compute a minimum cost $\tilde{b}_{x^*}$-edge cover $\bar{x}$ and output $\bar{x}$ as a solution to (7).

We show that algorithm COVER is an approximation algorithm for problem (7). We can see that $\bar{x}$ is a feasible for problem (7) since for each $S \in \mathcal{S}$, the vector $v = \arg\max_{u \in S} x^*(\delta(u))$ satisfies $\sum_{u \in S} \bar{x}(\delta(u)) \geq \bar{x}(\delta(v)) \geq \tilde{b}_{x^*}(v) \geq b(S)$. The approximation factor of algorithm COVER can be derived analogously to that of algorithm DOMINATE.

Lemma 5. *Let* $x \in \mathrm{DC}(G, b)$ *and* $h = \max_{S \in \mathcal{S}} |S|$. *Then vector* hx *satisfies conditions* (a) *and* (b) *for* $\mathrm{EC}(G, \tilde{b}_x)$, *where* $\tilde{b}_x \in \mathbb{Z}_+^V$ *is a vector obtained from* x *in Step 3 of* COVER.

Proof. By $x \in \mathrm{DC}(G, b)$, $x \in \mathbb{R}_+^E$ holds. Then vector hx satisfies (a) for $\mathrm{EC}(G, \tilde{b}_x)$. We show that hx satisfies (b), i.e., $hx(\delta(v)) \geq \tilde{b}_x(v)$ for each $v \in V$. Let v be a vertex in V. If $\tilde{b}_x(v) = 0$, then $hx(\delta(v)) \geq 0 = \tilde{b}_x(v)$ holds. Then assume $\tilde{b}_x(v) > 0$. There exists a subset $S \in \mathcal{S}$ such that $x(\delta(v)) \geq x(\delta(u))$ holds for all $u \in S$ and $\tilde{b}_x(v) = b'_x(v, S) = b(S)$. From this inequality and the condition $\sum_{u \in S} x(\delta(u)) \geq b(S)$ for $\mathrm{DC}(G, b)$, we have

$$hx(\delta(v)) \geq |S| x(\delta(v)) \geq \sum_{u \in S} x(\delta(u)) \geq b(S) = b'_x(v, S) = \tilde{b}_x(v).$$

This implies that hx satisfies (b) for $\mathrm{EC}(G, \tilde{b}_x)$. □

Lemmas 5 and 2 indicate the following theorem.

Theorem 7. *Algorithm* COVER *achieves the approximation factor of*

$$h \cdot \left(1 + \frac{1}{2\lfloor 3\beta/2 \rfloor + 1}\right) \left(\leq \frac{4}{3} h\right)$$

for problem (7), *where* $h = \max\{|S| \mid S \in \mathcal{S}\}$ *and* $\beta = \min_{S \in \mathcal{S}, b(S) \neq 0} b(S)$.

Proof. Omitted due to the limitation. □

References

1. E. M. Arkin, M. M. Halldórsson, and R. Hassin, Approximating the tree and tour cover of graph, *Information Processing Letters*, vol. 47, pp. 275–282, 1993.
2. R. Carr, T. Fujito, G. Konjevod, and O. Parekh, A $2\frac{1}{10}$-approximation algorithm for a generalization of the weighted edge-dominating set problem, In *Proceedings of the Eighth ESA*, pp. 132–142, 2000.
3. V. Chvátal, A greedy heuristics for the set covering problem, *Mathematics of Operations Research*, vol. 4, pp. 233–235, 1979.
4. T. Fujito and H. Nagamochi, A 2-approximation algorithm for the minimum weight edge dominating set problem, *Discrete Applied Mathmatics*, vol. 118, pp. 199–207, 2002.
5. M. R. Garey and D. S. Johnson, *Computers and Intractability; a Guide to the Theory of NP-completeness*, W. H. Freeman & Co., 1979.
6. F. Harary, *Graph Theory*, Addison-Wesley, 1969.
7. J. Könemann, G. Konjevod, O. Parekh, and A. Sinha, Improved approximations for tour and tree covers, *Algorithmica*, vol. 38, pp. 441–449, 2004.
8. L. Lovász, On the ratio of optimal integral and fractional covers, *Discrete Mathematics*, vol. 13, pp. 383–390, 1975.
9. K. G. Murty and C. Perin, A 1-matching blossom-type algorithm for edge covering problems, *Networks*, vol. 12, pp. 379–391, 1982.
10. A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer, 2003.
11. M. Yannakakis and F. Gavril, Edge dominating sets in graphs, *SIAM Journal on Applied Mathematics*, vol. 38, pp. 364–372, 1980.

Bounded Degree Closest k-Tree Power Is NP-Complete*

Michael Dom, Jiong Guo, and Rolf Niedermeier

Institut für Informatik, Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2, D-07743 Jena, Fed. Rep. of Germany
{dom,guo,niedermr}@minet.uni-jena.de

Abstract. An undirected graph $G = (V, E)$ is the k-power of an undirected tree $T = (V, E')$ if $(u, v) \in E$ iff u and v are connected by a path of length at most k in T. The tree T is called the tree root of G. Tree powers can be recognized in polynomial time. The thus naturally arising question is whether a graph G can be modified by adding or deleting a specified number of edges such that G becomes a tree power. This problem becomes NP-complete for $k \geq 2$. Strengthening this result, we answer the main open question of Tsukiji and Chen [COCOON 2004] by showing that the problem remains NP-complete when additionally demanding that the tree roots must have bounded degree.

1 Introduction

Root finding is a natural and well-studied problem in graph algorithmics (see [1, Section 10.6] and [10] for surveys). We call a graph $G' = (V', E')$ a *k-root* of a graph $G = (V, E)$ if $V' = V$ and there is an edge between vertices u and v in G iff there is a path of length at most k between u and v in G'. The other way round, G is the *k-power* of G'. Even determining whether a graph G possesses a 2-root is NP-complete [12].

Kearney and Corneil [8] directed the attention to a special case of the root finding problem by demanding G' to be a tree. Before that, Lin and Skiena [11] have already shown that it can be decided in linear time whether a graph is the 2-power of a tree. Kearney and Corneil generalized this result by showing that the tree root finding problem – called k-Tree Power problem – can be solved in polynomial time for any k. Moreover, they introduced an important generalization of root finding, yielding a natural graph modification problem. The question now is, given a graph G and a nonnegative integer ℓ, can G be modified by adding or deleting at most ℓ edges such that the resulting graph has a k-tree root. Call this problem Closest k-Tree Power. This "error correction scenario" takes into account that a graph might be close to being the k-power of a tree and one tries to find out how close it actually is by considering the number ℓ of edge modifications needed. Kearney and Corneil have shown that

* Supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 757–766, 2005.

the Closest k-Tree Power problem is NP-complete for $k \geq 3$. Moreover, it is reported that it is also NP-complete in the case $k = 2$ [7]. We strengthen these results to the case that the root trees may only have bounded degree.

Motivated by applications in computational biology, variants of k-Tree Power and Closest k-Tree Power have recently been studied [2, 13]. In these problems, only the leaves of the root are in one-to-one correspondence with the given graph vertices, the inner tree nodes are considered as "Steiner nodes" (see [2, 13] for details). The corresponding problems Closest k-Leaf Power and Closest k-Phylogenetic Power (where in the latter case all inner nodes of the tree have to have degree at least three) are NP-complete for $k \geq 2$. Intuitively speaking, these problems allow for a higher degree of freedom by freely choosing inner tree nodes and this may explain why, as opposed to tree root finding, polynomial-time solvability of the corresponding recognition problems k-Leaf Power and k-Phylogenetic Power is only known for $k \leq 4$ [2, 13]. The cases $k > 4$ are open in both settings. In addition, it has been strongly advocated to study the problems when the maximum node degree of the root tree is bounded from above by a constant [2, 3, 14]. In particular, Tsukiji and Chen [14] have proven that, for $k \geq 3$, the Closest k-Phylogenetic Power problem (called Closest k-Phylogenetic Root there) remains NP-complete when one demands that the root tree has bounded degree. The case $k = 2$ is open. Moreover, they emphasize that they leave open the "more fundamental" problem to determine the complexity of Closest k-Tree Power [14, page 461] in case of bounded degrees. They conjecture NP-completeness. We settle their open problem by proving this conjecture. More precisely, we show that Closest k-Tree Power is NP-complete for $k \geq 2$ and maximum node degree four in the root tree. We only leave open the case of maximum node degree three.

Let us briefly discuss our result. First, the NP-hardness proof of Kearney and Corneil [8] relies on the NP-completeness of the so-called Fitting Ultrametric Trees problem [9]. To show our result, we had to develop a completely different, more "fine-grained" sort of reduction from the NP-complete Vertex Cover for Graphs with Maximum Degree Three problem (3-Vertex Cover for short) [6]. Second, studying tree powers [8] instead of leaf powers [13] or phylogenetic powers [2], it is impossible to make use of the degree of freedom as provided by inner nodes in the latter two cases. Hence, NP-hardness appears to be harder to show here, somewhat explaining why the problem was left open and considered more fundamental in [14]. Using our new type of construction for the reduction, we could overcome this difficulty, improving Kearney and Corneil's construction [8] which makes use of unbounded degrees.

Due to the lack of space several proofs had to be omitted.

2 Preliminaries

We consider only undirected graphs $G = (V, E)$ with $n := |V|$ and $m := |E|$. Edges are denoted as tuples (u, v). The *degree* of a vertex v is the number of adjacent vertices. For a graph $G = (V, E)$ and $u, v \in V$, let $d_G(u, v)$ denote

the length of the shortest path between u and v in G. With $E(G)$, we denote the edge set of a graph G. We call a graph $G' = (V', E')$ an *induced subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' = \{(u, v) \mid u, v \in V' \text{ and } (u, v) \in E\}$. For two sets A and B, $A \triangle B$ denotes the *symmetric difference* $(A \setminus B) \cup (B \setminus A)$.

Given an unrooted tree T with node set V, the *k-tree power* of T is a graph, denoted by T^k with $T^k := (V, E)$, where $E := \{(u, v) \mid u, v \in V \text{ and } d_T(u, v) \leq k\}$. It can be decided in $O(n^3)$ time whether, for specified k, a graph is a k-tree power or not [8]. The more general graph modification problem that asks whether a given graph G is close to any k-tree power T^k then reads as follows.

Closest k-Tree Power (CTPk): Given a graph $G = (V, E)$ and a nonnegative integer ℓ, is there a tree T such that T^k and G differ by at most ℓ edges, that is $|E(T^k) \triangle E(G)| \leq \ell$? CTP$k$ is NP-complete for $k \geq 2$ [7, 8]. In this paper we study a special case of CTPk where the degree of every node in T is bounded from above by a fixed constant Δ.

Closest k-Tree Power with Maximum Degree Δ (Δ-CTPk): Given a graph $G = (V, E)$ and a nonnegative integer ℓ, is there a tree T with maximum node degree Δ such that T^k and G differ by at most ℓ edges, that is $|E(T^k) \triangle E(G)| \leq \ell$? Clearly, Δ-CTPk is in NP, because tree powers can be recognized in polynomial time [8]. It remains to show the NP-hardness.

Our reference point for showing NP-completeness of Δ-CTPk is 3-Vertex Cover: Given a graph $G = (V, E)$ with a maximum vertex degree 3 and a nonnegative integer ℓ, is there a set $C \subseteq V$ of at most ℓ vertices such that each edge from E has at least one endpoint in C? 3-Vertex Cover is NP-complete [6]. We show NP-completeness of Δ-CTPk for $k \geq 2$ and $\Delta \geq 4$ by proceeding as follows. First, we study the somewhat simpler case $k \geq 3$. Observe, however, that NP-completeness for some k does not immediately imply NP-completeness for $k + 1$. Second, we strengthen our findings by showing NP-completeness for $k = 2$ where some additional technical expenditure is needed.

To make the presentation clearer, we will speak of vertices when referring to a Vertex Cover input instance in the following sections and we will speak of nodes when referring to a Δ-CTPk instance.

3 Δ-CTPk Is NP-Complete for $k \geq 3$ and $\Delta \geq 4$

The central point in the NP-completeness proof is to "simulate" the Vertex Cover problem by the graph modification problem CTPk. In the course of this, we will ensure that a Vertex Cover input instance with maximum vertex degree three translates into an instance of CTPk with a desired tree root with maximum node degree four. In what follows, we briefly describe the fundamental ideas behind this reduction of 3-Vertex Cover to 4-CTPk.

Vertex covering means to find a minimum set of vertices that covers all edges. Equivalently, we may consider the following problem. Subdivide each edge of the graph into two edges by inserting a new vertex each time. Then, Vertex Cover can be seen as an edge deletion problem where the task is to break the graph into n connected components such that each connected component contains exactly

one original vertex. Moreover, one wants to maximize the number of connected components which consist of isolated vertices (or, equivalently, to minimize the number of connected components that contain at least one edge – the corresponding original vertices form the vertex cover). At first sight, this simply sounds as a rather complicated reformulation of Vertex Cover. The advantage is that this formulation is a step closer to our final goal, a graph modification problem where we modify edges.

So far, observe that in the "new" problem we always have to delete m edges to achieve n connected components as described above. Thus, one difficulty that remains to be solved is to interrelate the number of vertices in a vertex cover and the number of edges modified in 4-CTPk. In addition, we still have to bring into play the tree root problem as such. To this end, we make a construction as follows. Firstly, note that we will connect the n connected components described above by an additional "*backbone structure*" such that we finally can have a connected graph that has a tree root. Secondly, we employ an *edge gadget* which translates the edge deletion scenario of the reformulated Vertex Cover problem into an edge deletion *and* insertion scenario on the 4-CTPk side. It basically "expresses" that in vertex covering all edges have to be covered. Thirdly, we employ a *vertex gadget* which ensures that we have a one-to-one functional correspondence between the number of vertices of the 3-Vertex Cover instance and the number of edges to be deleted and inserted in the 4-CTPk instance. It basically expresses that we want to minimize the size of the vertex cover. More specifically, the 3-Vertex Cover problem has a solution of size ℓ iff the constructed 4-CTPk instance has a solution of size $3m+2\ell$. In fact, the first term comes from the edge gadgets, and the second term comes from the vertex gadgets. We illustrate our reduction by focussing on Δ-CTP3.

Construction of the Reduction. We now describe the details of the construction for Δ-CTP3 for $\Delta \geq 4$. Given an instance $G = (V, E)$ of 3-Vertex Cover with $V := \{v_1, \ldots, v_n\}$, we construct the graph $G_{\text{CTP}} = (V_{\text{CTP}}, E_{\text{CTP}})$ as follows.

For every vertex $v_i \in V$ there is a *vertex gadget* in G_{CTP} that contains a *vertex node* x_0^i, a *connection stub* consisting of eight nodes $x_1^i, \ldots, x_8^i$, an *edge stub* for every neighbor v_j of v_i – each edge stub consisting of two nodes $y_1^{i,j}$ and $y_2^{i,j}$ –, and edges as shown in Figure 1.

To build the mentioned backbone structure, we add $n-1$ *connection nodes* z^i with $1 \leq i < n$ to G_{CTP}, and for all $1 \leq i < n$ we insert edges between the vertex gadgets of v_i and v_{i+1} and between the gadgets and z^i as shown in Figure 2.

For each edge $(v_i, v_j) \in E$, we add to G_{CTP} an *edge node* $e^{i,j}$ and insert edges between $e^{i,j}$ and the nodes x_0^i, $y_2^{i,j}$, x_0^j, $y_2^{j,i}$ from the vertex gadgets of v_i and v_j. See Figure 3 for an illustration. We call $e^{i,j}$ together with the four edges incident to it the *edge gadget* for $(v_i, v_j) \in E$.

Clearly, G_{CTP} is not a 3-tree power if G contains any edges, because in a 3-tree power T^k that contains at least four nodes every node u has at least three pairwise connected neighbors. To see this, consider the 3-tree root T of T^k: If there is a node v with distance 3 from u in T, then the two vertices between u and v form a clique in T^k together with v and u. Similarly, one can also find

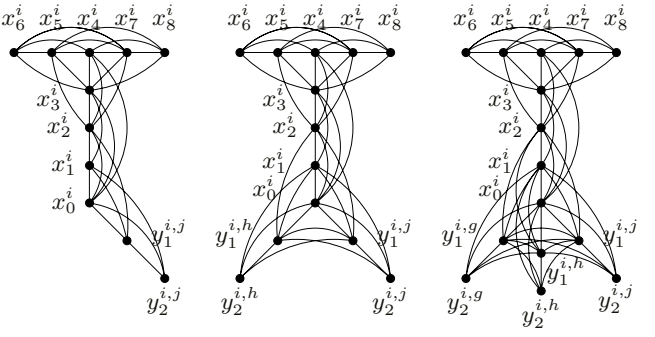


Fig. 1. The vertex gadget of a vertex $v_i \in V$. If v_i has only one neighbor v_j in G, the gadget of v_i has only one edge stub as shown on the left side. The illustrations in the middle resp. on the right side show the gadget of v_i in the case that v_i has two neighbors v_h, v_j resp. three neighbors v_g, v_h, v_j.

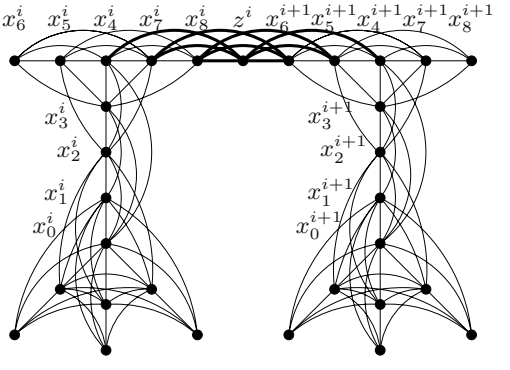


Fig. 2. The vertex gadgets of $v_i, v_{i+1} \in V$, and the connection node z^i. The edges inserted between the two gadgets and between the gadgets and z^i are drawn with bold lines.

three pairwise connected neighbors of u in T^k if all nodes in T are at distance at most 2 from u. However, every edge node $e^{i,j}$ in G_{CTP} has four neighbors with only two edges between them. Figure 4 gives an example for the reduction.

Correctness of the Reduction.

Proposition 1. *Let $G = (V, E)$ be an instance of* 3-Vertex Cover *and let G_{CTP} be the instance of Δ-CTP3 constructed from G as described above.*

If $C \subseteq V$ is a vertex cover for G, then G_{CTP} has a solution of size at most $3 \cdot m + 2 \cdot |C|$.

Proof. We prove the proposition by giving a solution of the postulated size for G_{CTP}. Let $C \subseteq V$ be a vertex cover for G, that is, every edge of E has at least one endpoint in C. Then we modify G_{CTP} as follows:

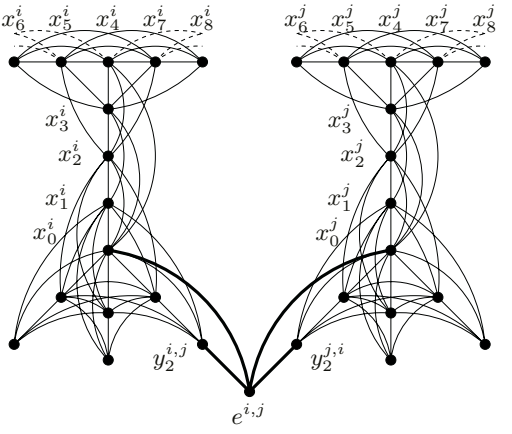


Fig. 3. The edge node $e^{i,j}$ for $(v_i, v_j) \in E$ and the vertex gadgets of v_i and v_j. The edges of the edge gadget are drawn with bold lines.

For every vertex $v_i \in C$ delete the edge (x_0^i, x_4^i) and insert the edge (x_1^i, x_4^i). For every edge $(v_i, v_j) \in E$, at least one of v_i and v_j, say v_i, is in C. Then insert the edge $(e^{i,j}, y_1^{i,j})$ and delete the edges $(e^{i,j}, x_0^j)$ and $(e^{j,i}, y_2^{j,i})$.

This solution has size $m \cdot 3 + |C| \cdot 2$, since we modify two edges in G_{CTP} for every vertex in C and three edges for every edge of E.

The resulting graph has a 3-tree root T with maximum vertex degree 4: Every edge node is connected with exactly one vertex gadget which is modified such that it has a 3-tree root as shown in Figure 4 for the gadget of v_2 (with x_1^2 lying between x_0^2 and x_2^2 in the tree root). If a vertex gadget is disconnected from all edge nodes, it has a 3-tree root like the gadgets of v_1 and v_3 in Figure 4 (with x_0^1 lying between x_1^1 and x_2^1). □

In order to show the reverse direction, we need the following lemma. We omit the lengthy proof.

Lemma 1. *Given a graph $G_{\text{CTP}} = (V_{\text{CTP}}, E_{\text{CTP}})$ constructed as described above, there is an optimal solution E_{opt} for Δ-CTP3 on G_{CTP} that leads to a graph $G_{\text{opt}} = (V_{\text{CTP}}, E_{\text{CTP}} \triangle E_{\text{opt}})$ with the following two properties:*
Edge node property. *Each edge node $e^{i,j}$ which is added into G_{CTP} for edge $(v_i, v_j) \in E$ has only three neighbors in G_{opt}, and these are either $x_0^i, y_1^{i,j}, y_2^{i,j}$ or $x_0^j, y_1^{j,i}, y_2^{j,i}$.*
Vertex node property. *For each vertex node x_0^i with $1 \leq i \leq n$, if there is an edge node $e^{i,j}$ adjacent to x_0^i in G_{opt}, then G_{opt} contains edge (x_1^i, x_4^i) but not edge (x_0^i, x_4^i); otherwise, G_{opt} contains (x_0^i, x_4^i) but not (x_1^i, x_4^i).*

Proposition 2. *Let $G = (V, E)$ be an instance of* 3-Vertex Cover *and let $G_{\text{CTP}} = (V_{\text{CTP}}, E_{\text{CTP}})$ be the instance of Δ-CTP3 constructed from G. If E_{sol} is a solution for G_{CTP}, then G has a vertex cover of size at most $(|E_{\text{sol}}| - 3 \cdot m)/2$.*

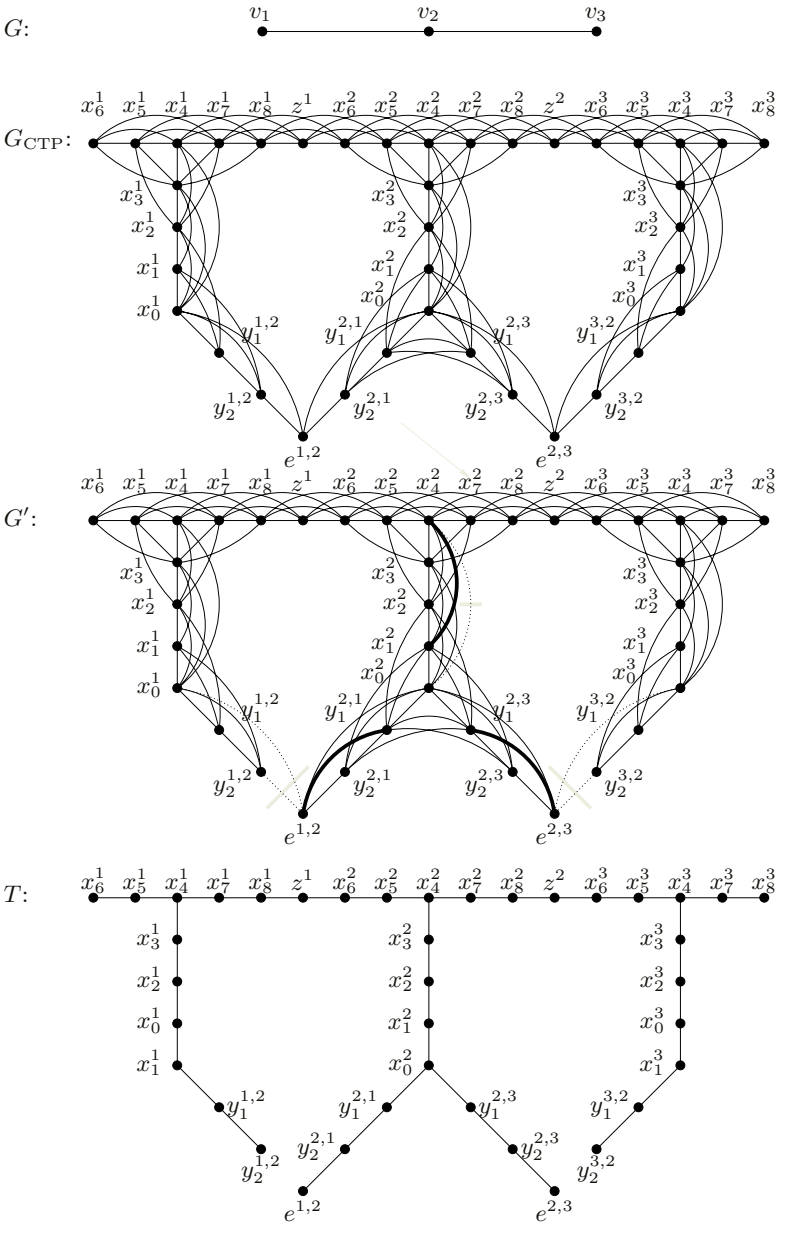


Fig. 4. An example reduction. Graph G is the 3-Vertex Cover instance, graph G_{CTP} is the graph constructed from G. There are three vertex gadgets with vertex nodes x_0^1, x_0^2, x_0^3 for the three vertices in G and two edge gadgets with edge nodes $e^{1,2}, e^{2,3}$ for the two edges of G. Graph G' results from G_{CTP} by deleting five edges and inserting three edges (inserted edges are drawn with bold lines, deleted edges with dotted lines). Graph G' is a 3-tree power with T as its 3-tree root. Note that we need three edge modifications for each edge of G and two edge modifications for the vertex gadget of v_2, which forms a vertex cover for G.

Proof. Let $E_{\text{sol}} \subseteq V_{\text{CTP}} \times V_{\text{CTP}}$ be an optimal solution for G_{CTP} as described in Lemma 1. In the resulting graph, every edge node $e^{i,j}$ is connected either to $x_0^i, y_1^{i,j}, y_2^{i,j}$ or to $x_0^j, y_1^{j,i}, y_2^{j,i}$ (edge node property of Lemma 1). Hence, for every edge node $e^{i,j}$ there are three edge modifications in E_{sol}, either the deletion of $(e^{i,j}, x_0^i)$, $(e^{i,j}, y_2^{i,j})$ and the insertion of $(e^{i,j}, y_1^{j,i})$, or the deletion of $(e^{i,j}, x_0^j)$, $(e^{i,j}, y_2^{j,i})$ and the insertion of $(e^{i,j}, y_1^{i,j})$.

As every edge node $e^{i,j}$ is adjacent to exactly one of the vertex nodes x_0^i and x_0^j in the resulting graph,

$$C := \{v_i \subseteq V \mid x_0^i \text{ is adjacent to at least one edge node}\}$$

is clearly a vertex cover for G. The vertex node property of Lemma 1 implies that for every vertex node x_0^i that is adjacent to at least one edge node, the solution E_{sol} contains two edge modifications, namely the deletion of (x_0^i, x_4^i) and the insertion of (x_1^i, x_4^i). Hence, there can be at most $(|E_{\text{sol}}| - 3 \cdot m)/2$ such vertex nodes, and the size of C, which consists of the corresponding vertices in V, is bounded from above by $(|E_{\text{sol}}| - 3 \cdot m)/2$. □

Theorem 1. *Δ-CTPk is* NP*-complete for $k = 3$ and $\Delta \geq 4$.*

To generalize to Δ-CTPk for $k > 3$, we use a straightforward extension of the construction used for the case $k = 3$. The gadget for a vertex $v_i \in V$ then consists of a vertex node x_0^i, $3k - 1$ nodes $x_1^i, \ldots, x_{3k-1}^i$ (the connection stub), and $k - 1$ nodes $y_1^{i,j}, \ldots, y_{k-1}^{i,j}$ for each neighbor v_j of v_i (the nodes of the edge stubs). For each edge $(v_i, v_j) \in E$ there is an edge node $e^{i,j}$ with edges to $x_0^i, y_{k-1}^{i,j}, x_0^j$ and $y_{k-1}^{j,i}$. All ideas and proofs used for $k = 3$ also hold for $k > 3$, which leads to the following theorem:

Theorem 2. *Δ-CTPk is* NP*-complete for $k > 3$ and $\Delta \geq 4$.*

4 Δ-CTP2 Is NP-Complete for $\Delta \geq 4$

In this section we show the NP-completeness of Δ-CTP2 for $\Delta \geq 4$. The reduction is also from 3-Vertex Cover. Compared to the reduction in Section 3, the only difference lies in the edge gadget. In the construction in Section 3, a decisive point was that in the edge gadget with edge node $e^{i,j}$ any optimal solution *needs* to disconnect exactly one of the two vertex gadgets corresponding to vertices v_i and v_j from $e^{i,j}$. More precisely, the vertex gadget for the covering vertex (or the vertex gadget for exactly one arbitrary covering vertex if there are two) stayed connected with $e^{i,j}$ whereas the vertex gadget for the other vertex became disconnected. In case of CTP2, however, with this construction it is no longer obvious that an optimal solution needs to disconnect exactly one of the two vertex gadgets. That is why we introduce a somewhat more complicated edge gadget, where we basically replace the *one* edge node $e^{i,j}$ by a clique of five nodes.

To present the refined construction and demonstrate its correctness, we employ forbidden subgraphs as shown in Figure 5. No 2-tree power has any of these as vertex-induced subgraph. A proof of the following lemma can be found in [5].

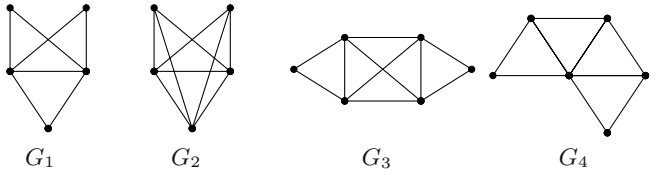


Fig. 5. Four forbidden induced subgraphs for 2-tree powers.

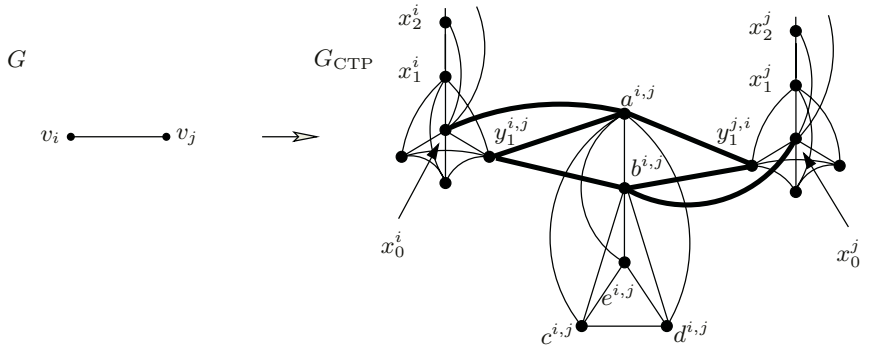


Fig. 6. The edge gadget used in the reduction from 3-Vertex Cover to Δ-CTP2 is a 5-nodes clique consisting of nodes $a^{i,j}, b^{i,j}, c^{i,j}, d^{i,j}, e^{i,j}$. The edges inserted between the edge gadget and the two vertex gadgets are drawn as bold lines. Nodes $y_1^{i,j}, y_1^{j,i}$ and $a^{i,j}, b^{i,j}$ together with each of $c^{i,j}, d^{i,j}, e^{i,j}$ form a forbidden induced subgraph G_1 as shown in Figure 5.

Lemma 2. *If a graph G has a 2-tree root, then G does not contain the subgraphs shown in Figure 5 as induced subgraphs.*

We use G_1 in Figure 5 to construct the edge gadget. The other three forbidden induced subgraphs are not directly used in the reduction but will be used in the proof of Lemma 3.

Since subgraph G_1 in Figure 5 is a forbidden induced subgraph for 2-tree powers, we need at least one edge modification to edit G_1 into a graph having a 2-tree root. Based on this observation, the edge gadget for $(v_i, v_j) \in E$ consists of five nodes which form a clique. Moreover, edges are inserted to connect two nodes of this edge gadget to the vertex gadgets of v_i and v_j to form induced subgraphs G_1, see Figure 6 for an illustration. Thus, if the 3-Vertex Cover instance G contains edges, then G_{CTP} is not a 2-tree power.

With exception of the edge gadget, the rest of the reduction is the same as the one in Section 3. The proof of the following lemma is very similar to the proofs of Propositions 1 and 2.

Lemma 3. *Given a* 3-Vertex Cover *instance $G = (V, E)$. Let G_{CTP} denote the graph constructed as described above. There is a vertex cover of ℓ vertices iff G_{CTP} can be transformed into a 2-tree power by $3 \cdot m + 2 \cdot \ell$ edge modifications.*

Theorem 3. *Δ-CTP2 is NP-complete for $\Delta \geq 4$.*

5 Conclusion

Showing NP-completeness of Closest k-Tree Power for $k \geq 2$ and maximum vertex degree four, we basically settled the open question of Tsukiji and Chen [14] and strengthened results of Kearney and Corneil [8]. Only the case with maximum vertex degree three is left open. We conjecture that by a further refinement of our type of reduction NP-completeness can be also shown here. Moreover, it would be interesting to study the complexities of the graph modification problems when one only allows either adding or deleting edges. Finally, investigating the polynomial-time approximability or fixed-parameter tractability of the proven NP-complete problems is a task for future research. Fixed-parameter tractability for closely related leaf root problems is shown in [4, 5].

References

1. A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: a Survey*. SIAM Monographs on Discrete Mathematics and Applications, 1999.
2. Z.-Z. Chen, T. Jiang, and G. Lin. Computing phylogenetic roots with bounded degrees and errors. *SIAM Journal on Computing*, 32(4):864–879, 2003.
3. Z.-Z. Chen and T. Tsukiji. Computing bounded-degree phylogenetic roots of disconnected graphs. In *Proc. 30th WG*, volume 3353 of *LNCS*, pages 308–319. Springer, 2004.
4. M. Dom, J. Guo, F. Hüffner, and R. Niedermeier. Error compensation in leaf root problems. In *Proc. 15th ISAAC*, volume 3341 of *LNCS*, pages 389–401. Springer, 2004. Long version to appear in *Algorithmica*.
5. M. Dom, J. Guo, F. Hüffner, and R. Niedermeier. Extending the tractability border for closest leaf powers. In *Proc. 31st WG*, LNCS. Springer, 2005. To appear.
6. M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.
7. T. Jiang, G. Lin, and J. Xu. On the closest tree kth root problem. Manuscript, Department of Computer Science, University of Waterloo, 2000.
8. P. E. Kearney and D. G. Corneil. Tree powers. *Journal of Algorithms*, 29(1):111–131, 1998.
9. M. Křivánek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986.
10. L. C. Lau and D. G. Corneil. Recognizing powers of proper interval, split, and chordal graphs. *SIAM Journal on Discrete Mathmatics*, 18(1):83–102, 2004.
11. Y. L. Lin and S. S. Skiena. Algorithms for square roots of graphs. *SIAM Journal on Discrete Mathematics*, 8(1):99–118, 1995.
12. R. Motwani and M. Sudan. Computing roots of graphs is hard. *Discrete Applied Mathematics*, 54(1):81–88, 1994.
13. N. Nishimura, P. Ragde, and D. M. Thilikos. On graph powers for leaf-labeled trees. *Journal of Algorithms*, 42(1):69–108, 2002.
14. T. Tsukiji and Z.-Z. Chen. Computing phylogenetic roots with bounded degrees and errors is hard. In *Proc. 10th COCOON*, volume 3106 of *LNCS*, pages 450–461. Springer, 2004.

A New Algorithm for the Hypergraph Transversal Problem*

Leonid Khachiyan[1], Endre Boros[2], Khaled Elbassioni[3], and Vladimir Gurvich[2]

[1] Department of Computer Science, Rutgers University
110 Frelinghuysen Road, Piscataway NJ 08854-8003
leonid@cs.rutgers.edu**
[2] RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003
{boros,gurvich}@rutcor.rutgers.edu
[3] Max-Planck-Institut für Informatik, Saarbrücken, Germany
elbassio@mpi-sb.mpg.de

Abstract. We consider the problem of finding all minimal transversals of a hypergraph $\mathcal{H} \subseteq 2^V$, given by an explicit list of its hyperedges. We give a new decomposition technique for solving the problem with the following advantages: (i) Global parallelism: for certain classes of hypergraphs, e.g. hypergraphs of bounded edge size, and any given integer k, the algorithm outputs k minimal transversals of $\mathcal{H}$ in time bounded by polylog$(|V|, |\mathcal{H}|, k)$ assuming poly$(|V|, |\mathcal{H}|, k)$ number of processors. Except for the case of graphs, none of the previously known algorithms for solving the same problem exhibit this feature. (ii) Using this technique, we also obtain new results on the complexity of generating minimal transversals for new classes of hypergraphs, namely hypergraphs of bounded dual-conformality, and hypergraphs in which every edge intersects every minimal transversal in a bounded number of vertices.

1 Introduction

Let $\mathcal{H} \subseteq 2^V$ be a hypergraph on a finite vertex set V of cardinality $|V| = n$. A vertex set $X \subseteq V$ is called a *transversal* of $\mathcal{H}$ if X intersects every hyperedge of $\mathcal{H}$. Let $\mathcal{H}^d = \{\text{minimal } X \subseteq V \mid X \text{ is a transversal to } \mathcal{H}\} \subseteq 2^V$ denote the set of all (inclusion-wise) minimal transversals of $\mathcal{H}$. We denote further by $\mathcal{H}^c \stackrel{\text{def}}{=} \{V \setminus H \mid H \in \mathcal{H}\}$ the *complementary* hypergraph of $\mathcal{H}$. We say that a hypergraph $\mathcal{H}$ is *Sperner* if no hyperedge of $\mathcal{H}$ contains another hyperedge. Clearly, $\mathcal{H}^{cc} = \mathcal{H}$, and it is not difficult to see that for Sperner hypergraphs $\mathcal{H}^{dd} = \mathcal{H}$. A vertex set $I \subseteq V$ is called an *independent* set of $\mathcal{H}$ if I contains no hyperedge of $\mathcal{H}$. Let $\mathcal{I}(\mathcal{H}) \subseteq 2^V$ denote the family of all maximal independent sets (MIS) of $\mathcal{H}$. Obviously, $I(\mathcal{H}) = \mathcal{H}^{dc}$. We assume that a Sperner hypergraph

* This research was supported in part by the National Science Foundation (NSF), grant IIS-0118635. The third author is also grateful for the partial support by DIMACS, the NSF's Center for Discrete Mathematics and Theoretical Computer Science.

** Our friend and co-author, Leonid Khachiyan passed away with tragic suddenness, while we were working on the final version of this paper.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 767–776, 2005.

$\mathcal{H}$ is given by the list of its hyperedges and consider problem $DUAL(\mathcal{H}, k)$ of generating k sets in $\mathcal{H}^d$ for a given integer k:
$DUAL(\mathcal{H}, k)$: *Given a Sperner hypergraph* $\mathcal{H} \subseteq 2^V$, *and an integer* $k \in \mathbb{Z}_+$ *output* $\min\{k, |\mathcal{H}^d|\}$ *minimal transversals (or equivalently, MIS's) of* $\mathcal{H}$.

This hypergraph transversal problem has received considerable attention in recent years [5, 7, 11–15, 23, 24] due to its generality and wide applicability in a variety of fields including artificial intelligence, reliability theory, database theory, integer programming, and learning theory (see, e.g. [6, 8, 13]). It is still open whether problem DUAL$(\mathcal{H}, k)$ can be solved in polynomial time for arbitrary hypergraphs. The fastest currently known algorithm [15] is quasi-polynomial and works by considering the following incremental generation problem:
$NEXT(\mathcal{H}, \mathcal{X})$: *Given a Sperner hypergraph* $\mathcal{H}$ *and a subset* $\mathcal{X} \subseteq \mathcal{H}^d$, *either find a new minimal transversal* $X \in \mathcal{H}^d \setminus \mathcal{X}$ *or show that* $\mathcal{X} = \mathcal{H}^d$.

The running time of the algorithm in [15] is $O(nN) + N^{o(\log N)}$, where $n = |V|$ and $N = |\mathcal{H}| + |\mathcal{X}|$. For several classes of hypergraphs, polynomial time algorithms exist, e.g. hypergraphs of bounded dimension (edge-size) [4, 7, 13], of bounded-degree [11, 14], of bounded-edge intersections [5], of bounded treewidth [14], and read-once (exact) hypergraphs [12].

In this paper, we consider a new decomposition method, for solving the hypergraph transversal problem, with the following advantages:
(i) *Global parallelism*: For hypergraphs of bounded dimension:

$$\dim(\mathcal{H}) \stackrel{\text{def}}{=} \max_{H \in \mathcal{H}} |H| < \delta, \tag{1}$$

for some constant $\delta \geq 2$, it was shown in [4], that problem NEXT$(\mathcal{H}, \mathcal{X})$ can be efficiently solved in parallel: NEXT$(\mathcal{H}, \mathcal{X}) \in NC$ for $\delta \leq 4$ and NEXT$(\mathcal{H}, \mathcal{X}) \in RNC$ for $\delta = 5, 6 \ldots$ Note, however, that this result implies only that we can solve problem DUAL$(\mathcal{H}, k)$, in parallel, in time $k \cdot \text{polylog}(n, |\mathcal{H}|, k)$ on poly$(n, |\mathcal{H}|, k)$ processors. Except for hypergraphs of dimension 2, that is for graphs [9], no *global parallel algorithms*, i.e. those that solve problem DUAL$(\mathcal{H}, k)$ in time polylog$(n, |\mathcal{H}|, k)$ on poly$(n, |\mathcal{H}|, k)$ processors, are known. As we shall see, our method allows for such global parallelism for any bounded-edge, as well as some other related classes of hypergraphs.
(ii) *Stronger results for new classes of hypergraphs*: Using the decomposition technique described in this paper, we also obtain stronger bounds on the (sequential) complexity of dualization for some classes of hypergraphs for which the only previously known result was the quasi-polynomial bound of [15]:

1. *Hypergraphs with bounded dual-conformality*: These are hypergraphs $\mathcal{H}$ for which the dual of $\mathcal{I}(\mathcal{H})$ has bounded-edge size (see e.g. [3]):

$$\dim(\mathcal{H}^{dcd}) < \delta, \tag{2}$$

for some constant $\delta \geq 2$. For instance, hypergraphs in which every two minimal transversals intersect in a bounded number of vertices, belong to this class. We show that it is possible to find all minimal transversals for such hypergraphs in polylog time using a polynomial number of processors.

2. *Hypergraphs in which every edge intersects every minimal transversal in a bounded number of vertices:*

$$|H \cap T| < \delta \text{ for every } H \in \mathcal{H} \text{ and every } T \in \mathcal{H}^d, \tag{3}$$

for some constant $\delta \geq 2$. These generalize read-once hypergraphs, in which every edge meets every minimal transversal in exactly one vertex (i.e. $\delta = 2$). Read-once hypergraphs were considered in [12], where a polynomial-delay algorithm was given. We show that, for any constant δ, we can find k minimal transversals of a hypergraph $\mathcal{H} \subseteq 2^V$ in this class in $\text{polylog}(n, |\mathcal{H}|, k) + \Delta \log |\mathcal{H}|$ time using a quasi-polynomial number of processors $\text{poly}(n, k, \Pi) \cdot |\mathcal{H}|^{O(\log |\mathcal{H}|)}$, where Δ and Π are respectively the parallel time and the number of processors required to generate a single minimal transversal of $\mathcal{H}$.

We describe the general decomposition method used for obtaining the above results in the next section. Direct application of this method yields our first result for hypergraphs of bounded edge-size, which will be presented in Section 3. Following that, we consider in Section 4 the class of hypergraphs of bounded dual-conformality and show how they can be dualized efficiently in parallel. Finally, a more involved application of this decomposition method gives us the claimed results about hypergraphs satisfying (3), and is presented in Section 5.

2 A Global Parallel Dualization Algorithm

Let V be a finite set. Given a hypergraph $\mathcal{H} \subseteq 2^V$ and a subset $S \subseteq V$, denote by $\mathcal{H}_S = \{H \in \mathcal{H} \mid H \subseteq S\}$ the sub-hypergraph of $\mathcal{H}$ induced by S. For $r \in \mathbb{Z}_+$ and $0 < \epsilon < 1$, denote by $\mathbb{H}(r, \epsilon)$ the family of hypergraphs $\mathcal{H} \subseteq 2^V$, such that for every $S \subseteq V$, there exist subsets $S_1, \ldots, S_r \subseteq S$ satisfying:

(H1) For every $H \in \mathcal{H}_S$, there exists an $i \in [r] = \{1, \ldots, r\}$ such that $S_i \supseteq H$.
(H2) $|S_i| \leq (1 - \epsilon)|S|$, for each $i \in [r]$ for which $|\mathcal{H}_{S_i}| > 1$.

Given two hypergraphs $\mathcal{H}_1, \mathcal{H}_2 \subseteq 2^V$, denote their conjunction by

$$\mathcal{H}_1 \bigwedge \mathcal{H}_2 = \{\text{minimal } H \mid H = H_1 \cup H_2 \text{ for some } H_1 \in \mathcal{H}_1 \text{ and } H_2 \in \mathcal{H}_2\}.$$

Proposition 1. *Let $\mathcal{H} \subseteq 2^V$ be a hypergraph and $S_1, \ldots, S_r \subseteq V$ be subsets such that for every $H \in \mathcal{H}$ there is an $i \in [r]$ such that $H \subseteq S_i$. Then $\mathcal{H}^d = \bigwedge_{i=1}^n \mathcal{H}^d_{S_i}$.*

Consider the following parallel algorithm for generating all minimal transversals of a hypergraph $\mathcal{H} \in \mathbb{H}(r, \epsilon)$:

Procedure DUALIZE$(\mathcal{H}, V)$**:**

1. If $|\mathcal{H}| = 0$, then return $\emptyset$.
2. If $\mathcal{H}$ has only one hyperedge H, then return $\{\{i\} \ : \ i \in H\}$.
3. In parallel, do the following:
4. find sets $S_1, \ldots, S_r \subseteq V$ satisfying (H1) and (H2) with $S = V$.
5. Let $\mathcal{G}_i \leftarrow$DUALIZE$(\mathcal{H}_{S_i}, S_i)$, for $i = 1, \ldots, r$.
6. Compute the conjunction $\mathcal{G} \leftarrow \bigwedge_{i=1}^r \mathcal{G}_i$.
7. Return $\mathcal{G}$.

Proposition 2 ([21]). *For any hypergraph $\mathcal{H} \subseteq 2^V$ and any $S \subseteq V$, we have $|\mathcal{H}_S^d| \leq |\mathcal{H}^d|$.*

Proposition 1 implies that the above procedure outputs all minimal transversals of $\mathcal{H} \in \mathbb{H}(r, \epsilon)$ correctly. The following proposition gives the running time and number of processors required by this procedure, in terms of the time $\tau = \tau(n, |\mathcal{H}|, r, \epsilon)$ and the number of processors $\pi = \pi(n, |\mathcal{H}|, r, \epsilon)$ needed to find the sets $S_1, \ldots, S_r \subseteq V$, satisfying (H1) and (H2). We shall assume in what follows that we run our procedures on a CREW-PRAM.

Proposition 3. *Let $t(n, m, M)$ and $p(n, m, M)$ be respectively the time and the number of processors, required by procedure DUALIZE$(\mathcal{H}, V)$ to output all minimal transversals of a hypergraph $\mathcal{H} \in \mathbb{H}(r, \epsilon)$ of size $m = |\mathcal{H}|$ on n vertices and with $|\mathcal{H}^d| = M$. Then $t(n, m, M) = O((\tau + \log(n(m + M^r))\frac{\log n}{\epsilon})$ and $p(n, m, M) = O((\pi + (m + M^{2r})n)n^{\frac{\log r}{\epsilon}})$.*

Note that, in the above procedure, all minimal transversals are generated simultaneously, at the very end, and there is no need to generate some of them individually, in the course of the computations. In other words, we did not need, in the above procedure, to compute a single minimal transversal in parallel. In general, the efficient parallel computation of a single minimal transversal of an arbitrary hypergraph is an outstanding open problem [18]. For some special classes of hypergraphs, e.g. hypergraphs of bounded dimension [1, 2, 10, 20], hypergraphs of bounded vertex-degrees [16], and linear hypergraphs [22], there exist efficient parallel algorithms for finding a minimal transversal.

However, if the requirement is not to generate all minimal transversals of $\mathcal{H}$ but only k of them, then the above procedure is not appropriate in its current form, but in fact can be modified, provided that we know how to generate an individual minimal transversal efficiently in parallel.

Procedure DUALIZE$(\mathcal{H}, V, k)$:

1. If $|\mathcal{H}| = 0$, then return $\emptyset$.
2. If $\mathcal{H}$ has only one hyperedge H, then return $\{\{i\} \ : \ i \in H\}$.
3. In parallel, do the following:
4. find sets $S_1, \ldots, S_r \subseteq V$ satisfying (H1) and (H2) with $S = V$.
5. Let $\mathcal{G}_i \leftarrow$DUALIZE$(\mathcal{H}_{S_i}, S_i, k)$, for $i = 1, \ldots, r$.
6. If there is an $i \in \{1, \ldots, r\}$ such that $|\mathcal{G}_i| = k$ then
7. In parallel, for each $Y \in \mathcal{G}_i$, do the following:
8. Let $\mathcal{H}[Y] = \{H \setminus S_i \mid H \cap Y = \emptyset\}$.
9. Compute a minimal transversal T_Y of $\mathcal{H}[Y]$.
10. Return $\mathcal{G} \leftarrow \{Y \cup T_Y \ : \ Y \in \mathcal{G}_i\}$, and stop.
11. else
12. Compute the conjunction $\mathcal{G} \leftarrow \bigwedge_{i=1}^{r} \mathcal{H}_{S_i}^d$.
13. Return $\min\{k, |\mathcal{G}|\}$ elements of $\mathcal{G}$.

Notation. In the rest of the paper, we use n, m respectively to denote the number of vertices $|V|$ and number of edges $|\mathcal{H}|$ of the input hypergraph $\mathcal{H} \subseteq 2^V$.

We denote respectively further by Δ and Π the parallel time and the number of processors required to generate a single minimal transversal for a hypergraph $\mathcal{H}$ belonging to the class under consideration.

Proposition 4. *DUALIZE*$(\mathcal{H}, V, k)$ *outputs* $\min\{k, |\mathcal{H}^d|\}$ *minimal transversals of a hypergraph* $\mathcal{H} \in \mathbb{H}(r, \epsilon)$ *in time* $t(n, m, k) = O((\tau + \log(n(m + k^r)) \log n/\epsilon + \Delta)$, *using* $p(n, m, k) = O((\pi + (m + k^{2r})n)n^{\log r/\epsilon} + k\Pi)$ *processors.*

Remark. More generally, we may consider a positive integer-valued, monotone set-function $f : V \mapsto \mathbb{Z}_+$, and replace condition (H2) above by

(H2′) $f(S_i) \leq (1 - \epsilon)f(S)$, for each $i \in \{1, \ldots, r\}$ for which $|\mathcal{H}_{S_i}| > 1$.

Then one can show similarly that the depth of the recursion tree required by procedure DUALIZE$(\mathcal{H}, V, k)$ is $O(\log f(V)/\epsilon)$. In Sections 3 and 4, we use $f(S) = |S|$. In Section 5, we use $f(S) = |\mathcal{H}_S|$.

3 Hypergraphs of Bounded Dimension

Let $\mathcal{H} \subseteq 2^V$ be a hypergraph on V, satisfying (1). As mentioned in the introduction, an efficient parallel algorithm exists [4] for solving problem NEXT$(\mathcal{H}, \mathcal{X})$ for such hypergraphs. This is based on an NC-reduction of problem NEXT$(\mathcal{H}, \mathcal{X})$ to problem DUAL$(\mathcal{H}', 1)$ of finding a single minimal transversal in a partial sub-hypergraph $\mathcal{H}'$ of $\mathcal{H}$, for which NC algorithms exist, if $\dim(\mathcal{H}) \leq 3$ ([1, 10]), and an RNC algorithm exists if $\dim(\mathcal{H}) = 4, 5, \ldots$ ([2, 20]). In particular, these reductions do not yield a global parallel algorithm. The case of graphs, $\dim(\mathcal{H}) \leq 2$, was considered in [9], where it was shown that problem DUAL$(\mathcal{H}, k)$ can be solved in $O(\log^3(nk))$ parallel time on $O(n^2k^6)$ processors, on a CREW-PRAM. Here we show the following stronger and more general results.

Theorem 1. *There is a deterministic parallel algorithm that, for any hypergraph* $\mathcal{H} \subseteq 2^V$ *such that* $\dim(\mathcal{H}) < \delta$, *solves problem DUAL*$(\mathcal{H}, |\mathcal{H}^d|)$ *in time* $O(\delta^2 \log n \log(n|\mathcal{H}^d|))$ *using* $O(n^{\delta \log \delta + 1}(m + |\mathcal{H}^d|^{2\delta}))$ *processors.*

We also get an NC reduction of finding k minimal transversals of $\mathcal{H}$, when $\dim(\mathcal{H}) \leq Const.$, to computing a single minimal transversal in a restricted sub-hypergraph of $\mathcal{H}$.

Theorem 2. *There is a deterministic parallel algorithm that, for any* $\mathcal{H} \subseteq 2^V$ *such that* $\dim(\mathcal{H}) < \delta$, *and any integer* $k \in \mathbb{Z}_+$, *solves problem DUAL*$(\mathcal{H}, k)$ *in time* $O(\delta^2 \log n \log(nk) + \Delta)$ *using* $O(n^{\delta \log \delta + 1}(m + k^{2\delta}) + k\Pi)$ *processors.*

Theorem 2 extends the results of [2, 20] which show that problem DUAL$(\mathcal{H}, k) \in RNC$, for hypergraphs $\mathcal{H}$ of bounded dimension and for $k = 1$, to any integer k. Theorem 2 also extends the results of [9] where it was shown that problem DUAL$(\mathcal{H}, k) \in NC$, for hypergraphs of dimension 2 and for any integer k, to hypergraphs of any bounded dimension.

Proofs of Theorems 1 and 2. We apply the results of the preceding section to solve problem DUAL$(\mathcal{H}, k)$, by setting $r = \delta$ and $\epsilon = 1/\delta$. Then $\mathcal{H} \in \mathbb{H}(r, \epsilon)$. Indeed, given any set $S \subseteq V$, we partition S into r (almost) equal parts $W_1, \ldots, W_r$, and let $S_i = S \setminus W_i$, for $i = 1, \ldots, r$. The fact that any hyperedge $H \in \mathcal{H}_S$ has size at most $\delta - 1$ implies that H cannot intersect all sets $W_1, \ldots, W_r$, i.e. H must be contained in at least one of the sets $S_1, \ldots, S_r$. This implies that (H1) holds for $\mathcal{H}$. (H2) follows form the fact that $|S_i|$ is roughly $\frac{(\delta-1)|S|}{\delta}$, for $i = 1, \ldots, r$. Thus Theorems 1 and 2 follow from Propositions 3 and 4 respectively. □

4 Hypergraphs of Bounded Dual-Conformality

Given a hypergraph $\mathcal{H} \subseteq 2^V$, a vertex set $S \subseteq V$ is called a *sub-transversal* of $\mathcal{H}$ if $S \subseteq X$ for some minimal transversal $X \in \mathcal{H}^d$. By the above definitions, the hypergraph $\mathcal{H}^{dcd}$ is just the family of *minimal non sub-transversals* of $\mathcal{H}$. For a subset $S \subseteq V$, and a vertex $v \in S$, let $\mathcal{H}_v(S) = \{H \in \mathcal{H} \mid H \cap S = \{v\}\}$ and let $\mathcal{H}_0(S) = \{H \in \mathcal{H} \mid H \cap S = \emptyset\}$. A selection of $|S|$ hyperedges $\{H_v \in \mathcal{H}_v(S) \mid v \in S\}$ is called *covering* if there exists a hyperedge $H \in \mathcal{H}_0(S)$ such that $H \subseteq \bigcup_{v \in S} H_v$. The next proposition states that a non-empty set S is a sub-transversal of $\mathcal{H}$ if and only if there exists a non-covering selection for S.

Proposition 5 (cf. [7]). *Let $S \subseteq V$ be a non-empty vertex set in a hypergraph $\mathcal{H} \in 2^V$.*

i) *If S is a sub-transversal for $\mathcal{H}$ then there exists a non-covering selection $\{H_v \in \mathcal{H}_v(S) \mid v \in S\}$ for S.*

ii) *Given a non-covering selection $\{A_v \in \mathcal{A}_v(S) \mid v \in S\}$ for S, we can extend S to a minimal transversal of $\mathcal{H}$ by solving problem DUAL$(\mathcal{H}', 1)$ for the induced partial hypergraph $\mathcal{H}' = \{H \cap U \mid H \in \mathcal{H}_0(S)\} \subseteq 2^U$, where $U = V \setminus \bigcup_{v \in S} H_v$.*

In general, finding a non-covering selection for S (or equivalently, testing if S is a sub-transversal) is NP-hard if the cardinality of S is not bounded. In fact, this is so even for $\dim(\mathcal{H}) = 2$, that is for graphs (see [4]). However, if the size of S is bounded by a constant then there are only polynomially many selections $\{H_v \in \mathcal{H}_v(S) \mid v \in S\}$ for S. All of these selections, including the non-covering ones, can be enumerated efficiently in parallel.

Corollary 1. *For any fixed δ there is an algorithm which, given a hypergraph $\mathcal{H} \subseteq 2^V$ and a set S of less than δ vertices, determines whether S is a sub-transversal to $\mathcal{H}$ and if so finds a non-covering selection $\{H_v \in \mathcal{H}_v(S) \mid v \in S\}$, in $O(\log(nm))$ parallel time using $O(nm^\delta)$ processors.*

Let $\mathcal{H}$ be a hypergraph such that (2) is satisfied for some constant δ. It is not clear how to check (2), even for $\delta = 3$. However, as far as the generation of the dual hypergraph $\mathcal{H}^d$ is concerned, such a check is not needed. In fact, we present below an efficient parallel algorithm that, for any given constant δ, either solves problem DUAL$(\mathcal{H}, k)$ or discovers that (2) is not satisfied.

Theorem 3. *There is a deterministic parallel algorithm that, given $\mathcal{H} \subseteq 2^V$, $k \in \mathbb{Z}_+$, and a constant integer δ, either solves problem DUAL$(\mathcal{H}, k)$, or proves that $\dim(\mathcal{H}^{dcd}) \geq \delta$, in time $O(\log n \log(nmk) + \Delta)$ using $O((nmk)^{2\delta \log \delta} + k\Pi)$ processors, where Δ and Π are respectively the parallel time and the number of processors required to generate a single minimal transversal in a hypergraph of dimension less than δ.*

A hypergraph $\mathcal{H} \subseteq 2^V$ is said to be δ-*conformal* [3] if for every vertex-set $X \subseteq V$ the following property holds: X is contained in a hyperedge of $\mathcal{H}$ whenever each subset of X of cardinality at most δ is contained in a hyperedge of $\mathcal{H}$. It is not difficult to see that a hypergraph $\mathcal{H}$ satisfies (2) if and only if its dual $\mathcal{H}^d$ is $(\delta-1)$-conformal. Note that the dualization problem for hypergraphs $\mathcal{H}$ satisfying $\dim(\mathcal{H}^d) < \delta$, for some constant δ, can be trivially solved efficiently in parallel, just by enumerating all subsets of vertices of size less than δ, and checking which of them are minimal transversals. Note further that $\dim(\mathcal{H}^d) \leq \delta$ implies that $\dim(\mathcal{H}^{dcd}) \leq \delta+1$. Thus Theorem 3 extends the parallel dualization result for hypergraphs whose duals are of bounded dimension to the wider class of hypergraphs whose duals are δ-conformal, for some constant δ.

Proof of Theorem 3. We present an algorithm that, will keep generating in parallel minimal transversals of $\mathcal{H}$, and halt only when either all or at least k of such transversals have been generated, or when the algorithm discovers that condition (2) is not satisfied. The algorithm proceeds in the following two steps:

Step 1. Generate the hypergraph $\mathcal{F} \subseteq 2^V$, whose hyperedges are defined as follows: $\mathcal{F} = \{S \subseteq V \ : \ |S| < \delta$ and S is a minimal non sub-transversal of $\mathcal{H}\}$. For a constant δ, the hypergraph $\mathcal{F}$ can be generated in $O(\log(nm))$ parallel time using $O((nm)^\delta)$ processors by Corollary 1.

Step 2. Note that $\dim(\mathcal{F}) < \delta$. Thus Theorem 2 implies that problem DUAL$(\mathcal{F}, k)$ can be solved efficiently in global *RNC* time. However, we may not need to generate all the hyperedges of $\mathcal{F}^d$. We stop generation when either an edge $X \in \mathcal{F}^d$ is generated such that $V \setminus X$ is not a minimal transversal of $\mathcal{H}$, or when k edges of $\mathcal{F}^d$ have been generated, whichever happens sooner.

To verify that the above procedure indeed generates k transversals of $\mathcal{H}^d$ in global *RNC* time if (2) is satisfied, notice the equivalences

$$\dim(\mathcal{H}^{dcd}) \leq \delta \iff \mathcal{F} = \mathcal{H}^{dcd} \iff \mathcal{F}^{dc} \subseteq \mathcal{H}^d.$$

Now Theorem 3 becomes a consequence of Theorem 2, applied to the hypergraph $\mathcal{F}$ constructed in Step 1 above. □

5 Hypergraphs Generalizing Read-Once Hypergraphs

Let V be a finite set, $\delta \geq 2$ be a positive constant, and $\mathcal{H} \subseteq 2^V$ be a hypergraph satisfying (3). Note that testing a hypergraph $\mathcal{H}$ for (3) can be done efficiently in parallel. Indeed, $\mathcal{H}$ satisfies (3) if and only if for every edge $H \in \mathcal{H}$ and every subset $X \subseteq H$ of size $|X| = \delta$, X is not a sub-transversal to $\mathcal{H}$.

If $\delta = 2$, any hypergraph satisfying (3) is read-once (exact). For such hypergraphs, the following corollary is almost immediate from Theorem 1.

Corollary 2. *Let $\mathcal{H} \subseteq 2^V$ be a read-once hypergraph. Then for any integer k Problem DUAL$(\mathcal{H}, k)$ can be solved deterministically in polylog$(|V|, |\mathcal{H}|, k)$ parallel time using* poly$(|V|, |\mathcal{H}|, k)$ *number of processors.*

It is not known whether a similar criterion can be used to reduce the dualization of hypergraphs satisfying (3) to that of hypergraphs of bounded dimension. Nevertheless, we use our decomposition approach to prove the following result.

Theorem 4. *For any hypergraph $\mathcal{H} \subseteq 2^V$ satisfying (3), Computing k elements of $\mathcal{H}^d$ can be done in $O((\delta \log(nmk) + \Delta)\delta \log m)$ parallel time using $O(((nmk^2)^\delta + k\Pi)m^{2\delta^2 \log m})$ processors.*

Proof. Let $\mathcal{H}$ be a hypergraph satisfying (3). For a subset $S \subseteq V$, let $\mathcal{H}(S) = \{H \in \mathcal{H} \ : \ H \cap S \neq \emptyset\}$, $\mathcal{H}^S = \{\text{minimal } (H \cap S) \mid H \in \mathcal{H},\ H \cap S \neq \emptyset\}$ and $\mathcal{H}_S = \{H \in \mathcal{H} \mid H \subseteq S\}$. Note that if $\mathcal{H}$ satisfies (3) then so does $\mathcal{H}_S$ for any $S \subseteq V$, while $\mathcal{H}^S$ satisfies (3) if S is a transversal to $\mathcal{H}$. Denote by $\deg(v) = \deg_{\mathcal{H}}(v)$ the number of hyperedges of $\mathcal{H}$ containing $v \in V$, and let $0 < \epsilon_1 < \epsilon_2 < 1/(\delta - 1)$ be positive constants. Let $\epsilon = \max\{1 - \epsilon_1, (\delta - 1)\epsilon_2, 1 - (\epsilon_2 - \epsilon_1)\} \in (0, 1)$. To generate k minimal transversals of $\mathcal{H}$, we use the following procedure:
Step 1. Set $Z \leftarrow \cup\{H \in \mathcal{H} \ : \ |H| = 1\}$, $V \leftarrow V \setminus Z$, and $\mathcal{H} \leftarrow \mathcal{H}_V$.
Step 2. If $|\mathcal{H}| = 0$, then return $\{Z\}$. If $\mathcal{H}$ has only one hyperedge H, then return $\{Z \cup \{i\} \ : \ i \in H\}$.
Step 3. Let $L = \{v \in V \ : \ \deg(v) \geq \epsilon_1|\mathcal{H}|\}$. If $V \setminus L$ is a transversal to $\mathcal{H}$, then we proceed with Steps 3.1, 3.2, and 3.3 below, otherwise, we go to Step 4.
Step 3.1. Let $T = \{v_1, \ldots, v_t\} \subseteq V \setminus L$ be a minimal transversal to $\mathcal{H}$. Let $T_1 \cup T_2 \cup \ldots \cup T_\delta = T$ be a partition of T computed by the following procedure: (a) find indices $i_1, \ldots, i_{\delta-1} \in [t]$ such that, for $j = 1, \ldots, \delta - 1$,

$$|\mathcal{H}(\{v_{i_{j-1}+1}, \ldots, v_{i_j}\})| \leq \epsilon_2|\mathcal{H}| \text{ and } |\mathcal{H}(\{v_{i_{j-1}+1}, \ldots, v_{i_j+1}\})| > \epsilon_2|\mathcal{H}|,$$

where $i_0 = 0$ and $i_\delta = t$, by definition (b) set $T_j \leftarrow \{v_{i_{j-1}+1}, \ldots, v_{i_j}\}$ and $S_j \leftarrow V \setminus T_j$, for $j = 1, \ldots, \delta$. Note that for every $H \in \mathcal{H}$, since $|H \cap T| \leq \delta - 1$, we have $H \subseteq S_j$ for some $j \in [\delta]$. Note also that $|\mathcal{H}_{S_j}| < (1 - (\epsilon_2 - \epsilon_1))|\mathcal{H}|$, for $j = 1, \ldots, \delta$, and $|\mathcal{H}_{S_\delta}| \leq (\delta - 1)\epsilon_2|\mathcal{H}|$.
Step 3.2. Let recursively (and in parallel) $\mathcal{G}_j \leftarrow DUAL(\mathcal{H}_{S_j}, k)$, for $j = 1, \ldots, \delta$.
Step 3.3. If $|\mathcal{G}_j| = k$, for some $j \in [\delta]$, then compute (in parallel) for each $Y \in \mathcal{G}_j$ a minimal transversal T_Y of the sub-hypergraph $\mathcal{H}[Y] = \{H \setminus S_j \ : \ H \cap Y = \emptyset\} \subseteq 2^{V \setminus V_j}$. Note that $\mathcal{H}[Y]$ also satisfies (3). Return $\mathcal{G} \leftarrow \{Y \cup T_Y \cup Z \ : \ Y \in \mathcal{G}_j\}$.
Step 3.4. Otherwise, return k elements of the conjunction $\mathcal{G} \leftarrow \mathcal{G}_1 \wedge \ldots \wedge \mathcal{G}_\delta \wedge \{Z\}$.
Step 4. If there are distinct vertices $v_1, \ldots, v_\delta \in L$ such that no edge of $\mathcal{H}$ contains $\{v_1, \ldots, v_\delta\}$, then proceed with Steps 4.1 and 4.2 below, otherwise go to Step 5.
Step 4.1. Let $S_j \leftarrow V \setminus \{v_j\}$, for $j = 1, \ldots, \delta$. Then $\mathcal{H} = \bigcup_{j=1}^{\delta} \mathcal{H}_{S_j}$, and $|\mathcal{H}_{S_j}| \leq (1 - \epsilon_1)|\mathcal{H}| \leq \epsilon|\mathcal{H}|$, for $j = 1, \ldots, \delta$.

Step 4.2. Compute recursively $\mathcal{G}_j \leftarrow DUAL(\mathcal{H}_{S_j}, k)$, for $j = 1, \ldots, \delta$, and continue as in Steps 3.3 and 3.4 above.
Step 5. Assume now that $V \setminus L$ is not a transversal to $\mathcal{H}$, and that for every subset $L' \subseteq L$ of δ distinct vertices, there is an edge $H \in \mathcal{H}$ such that $H \supseteq L'$. Then for every minimal transversal $T \in \mathcal{H}^d$, we have $1 \leq |T \cap L| \leq \delta - 1$ by (3).

In this case, we proceed as follows. Let $\mathcal{S} \stackrel{\text{def}}{=} \{S \subseteq L \mid \exists T \in \mathcal{H}^d : T \cap L = S\}$, and for $S \in \mathcal{S}$, denote by V_S the set $V \setminus (L \setminus S)$. Elements of $\mathcal{S}$ can be identified as follows: a non-empty set $S \subseteq L$ of size at most $\delta - 1$ is in $\mathcal{S}$ if and only if (i) V_S is a transversal to $\mathcal{H}$, and (ii) S is a sub-transversal of the hypergraph $\mathcal{H}^{V_S}$. For $S \in \mathcal{S}$, let $\mathcal{T}(S) = \{T \in \mathcal{H}^d \mid T \cap L = S\}$. Then $\mathcal{H}^d$ is the disjoint union

$$\mathcal{H}^d = \left(\bigcup_{S \in \mathcal{S}} \mathcal{T}(S) \right) \wedge \{Z\}. \tag{4}$$

For each $S \in \mathcal{S}$, we find $\mathcal{T}(S)$ by applying the sub-transversal criterion (Proposition 5). More precisely, for $v \in S$, let $\mathcal{H}_v(S) = \{H \in \mathcal{H}^{V_S} \mid H \cap S = \{v\}\}$ and $\mathcal{H}_0(S) = \{H \in \mathcal{H}^{V_S} \mid H \cap S = \emptyset\}$. Let $\mathbb{F}_S$ be the family of non-covering selections of $\mathcal{H}^{V_S}$ with respect to S, i.e. collections of $|S|$ hyperedges $\{H_v \in \mathcal{H}_v(S) \mid v \in S\}$ for which there exists no hyperedge $H \in \mathcal{H}_0(S)$ with $H \subseteq \bigcup_{v \in S} H_v$. For every $\mathcal{F} = \{H_v \in \mathcal{H}_v(S) \mid v \in S\} \in \mathbb{F}_S$, denote by $V_{\mathcal{F}} = V \setminus \left(\bigcup_{H \in \mathcal{F}} H \cup L \right)$. Then it is easy to see that

$$\mathcal{T}(S) = \bigcup_{\mathcal{F} \in \mathbb{F}_S} (\mathcal{H}_0(S)^{V_{\mathcal{F}}})^d \wedge \{S\}. \tag{5}$$

Note that, for all $\mathcal{F} \in \mathbb{F}_S$ and $S \in \mathcal{S}$, we have $|\mathcal{H}_0(S)^{V_{\mathcal{F}}}| \leq (1 - \epsilon_1)|\mathcal{H}|$ since none of the edges intersecting S belongs to $\mathcal{H}_0(S)$. Using (4) and (5), we compute k elements of $\mathcal{H}^d$ by recursively finding $\mathcal{G}_{\mathcal{F}} \leftarrow DUAL(\mathcal{H}_0(S)^{V_{\mathcal{F}}}, k)$ for every $\mathcal{F} \in \mathbb{F}_S$ and $S \in \mathcal{S}$. We stop either when $|\mathcal{G}_{\mathcal{F}}| = k$ for some $\mathcal{F} \in \mathbb{F}_S$ and some $S \in \mathcal{S}$, or when all families $(\mathcal{H}_0(S)^{V_{\mathcal{F}}})^d$ have been generated, for all $\mathcal{F} \in \mathbb{F}_S$ and $S \in \mathcal{S}$, in which case we use (4) and (5) to compute $\mathcal{H}^d$.

This completes our procedure for finding $\mathcal{H}^d$. The following proposition gives the parallel running time $t(n, m, k)$ and the number of processors $p(n, m, k)$ required to generate k minimal transversals, in terms of n, m, and k.

Proposition 6. *$t(n, m, k) = O((\delta \log(nmk) + \Delta) \log m / \log \frac{1}{\epsilon})$ and $p(n, m, k) = O(((nmk^2)^\delta + k\Pi) m^{\delta \log m / \log \frac{1}{\epsilon}})$.*

Setting $\epsilon = 1 - 1/(2\delta)$ in Proposition 6 completes the proof of Theorem 4. □

References

1. N. Alon, L. Babai, A. Itai, A fast randomized parallel algorithm for the maximal independent set problem, J. Algorithms 7 (1986) pp. 567–583.
2. P. Beame, M. Luby, Parallel search for maximal independence given minimal dependence, in *Proc. of the First SODA Conference (1990)*, pp. 212–218.

3. C. Berge, *Hypergraphs*, North Holland Mathematical Library, Vol. 445, 1989.
4. E. Boros, K. Elbassioni, V. Gurvich, and L. Khachiyan, An efficient incremental algorithm for generating all maximal independent sets in hypergraphs of bounded dimension, *Parallel Processing Letters*, 10 (2000), pp. 253–266.
5. E. Boros, K. Elbassioni, V. Gurvich and L. Khachiyan, Generating Maximal Independent Sets for Hypergraphs with Bounded Edge-Intersections, in *Proc. 6th Latin American Theoretical Informatics Conference* 2004, *LNCS* 2976, pp. 488–498.
6. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan and K.Makino, Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities, *SIAM J. Comput.*, 31(5) (2002) pp. 1624–1643.
7. E. Boros, V. Gurvich, and P.L. Hammer, Dual subimplicants of positive Boolean functions, *Optimization Methods and Software*, 10 (1998) pp. 147–156.
8. C. J. Colbourn, *The combinatorics of network reliability*, Oxford Univ. Press, 1987.
9. E. Dahlhaus and M. Karpinski, A fast parallel algorithm for computing all maximal cliques in a graph and the related problems, *Proc. 1st Scandinavian Workshop on Algorithm Theory (SWAT)*, Sweden, July 5-8, 1988, LNCS 318, pp. 139–144.
10. E. Dahlhaus, M. Karpinski and P. Kelsen, An efficient parallel algorithm for computing a maximal independent set in a hypergraph of dimension 3, *Inf. Process. Lett.* 42(6) (1992), pp. 309–313.
11. C. Domingo, N. Mishra and L. Pitt, Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries, *Machine learning* 37 (1999) pp. 89–110.
12. T. Eiter, Exact Transversal Hypergraphs and Application to Boolean μ-Functions, *J. Symb. Comput.* 17(3) (1994), pp. 215–225.
13. T. Eiter and G. Gottlob, Identifying the minimal transversals of a hypergraph and related problems, *SIAM J. Comput.*, 24 (1995), pp. 1278–1304.
14. T. Eiter, G. Gottlob and K. Makino, New results on monotone dualization and generating hypergraph transversals, in *Proc. 34-th Anual ACM STOC Conf.*, 2002, pp. 14–22.
15. M. L. Fredman and L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms*, 21 (1996) pp. 618–628.
16. O. Garrido, P. Kelsen and A. Lingas, A simple NC-algorithm for a maximal independent set in a hypergraph of polylog arboricity, *Information Processing Letters* 58(2) (1996), pp. 55–58.
17. D. S. Johnson, M. Yannakakis and C. H. Papadimitriou, On generating all maximal independent sets, *Information Processing Letters*, 27 (1988), pp. 119–123.
18. R. Karp and A. Wigderson, A fast parallel algorithm for the maximal independent set problem, JACM 32 (1985) pp. 762–773.
19. R. Karp, E. Upfal, and A. Wigderson, The complexity of parallel search, *Journal of Computer and System Science*, 36 (1988) pp. 225–253.
20. P. Kelsen, On the parallel complexity of computing a maximal independent set in a hypergraph, *Proc. 24-th Anual ACM STOC Conf.* (1992).
21. E. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, *SIAM J. Comput.*, 9 (1980) pp. 558–565.
22. T. Luczak and E. Szymanska, A parallel randomized algorithm for finding a maximal independent set in a linear hypergraph, *J. Algorithms* 25(2) (1997), 311–320.
23. K. Makino, Efficient Dualization of $O(\log n)$-Term Monotone Disjunctive Normal Forms, *Discrete Applied Mathematics*, 126 (2003) pp. 305–312.
24. H. Tamaki, Space-efficient enumeration of minimal transversals of a hypergraph, IPSJ-AL 75 (2000), pp. 29–36.

On Finding a Shortest Path in Circulant Graphs with Two Jumps

Domingo Gómez, Jaime Gutierrez, Álvar Ibeas,
Carmen Martínez, and Ramón Beivide

Faculty of Sciences, University of Cantabria
Santander E–39071, Spain
jaime.gutierrez@unican.es

Abstract. In this paper we present algorithms for finding a shortest path between two vertices of any weighted undirected and directed circulant graph with two jumps. Our shortest path algorithm only requires $O(\log N)$ arithmetic steps and the total bit complexity is $O(\log^3 N)$, where N is the number of the graph's vertices. This method has been derived from some Closest Vector Problems (CVP) of lattices in dimension two and with ℓ_1-norm.

1 Introduction

An *undirected circulant graph* $C_N(j_1, j_2, \ldots, j_m)$ with N vertices, labeled with integers modulo N, and jumps $j_1, j_2, \ldots, j_m$, is a graph in which each vertex n, $0 \leq n \leq N-1$, is adjacent to all the vertices $n \pm j_i \ mod\ N$, with $1 \leq i \leq m$. In contrast, a *directed circulant graph* $DC_N(j_1, j_2, \ldots, j_m)$ with N vertices, and jumps $j_1, j_2, \ldots, j_m$ is a graph in which each vertex n, $0 \leq n \leq N-1$, is adjacent to all the vertices $n+j_i \ mod\ N$, with $1 \leq i \leq m$. Throughout the paper we employ the term circulant graph for both undirected and directed circulant graphs.

This kind of graphs have a vast number of applications in telecommunication networking, VLSI design and distributed computation. Their properties, such as diameters and reliabilities, have been the focus of many research in computer network design [1–3, 6, 15, 19].

Every circulant graph can be associated to a lattice $\mathcal{L}$ which consists of the integer solutions $(x_1, \ldots, x_m) \in \mathbb{Z}^m$ to the system of congruences

$$j_1 x_1 + \cdots + j_m x_m \equiv 0 \bmod N. \tag{1}$$

Given two vertices r and s, a path from r to s in $C_N(j_1, j_2, \ldots, j_m)$ can be described by an integer vector $\mathbf{x} = (x_1, \ldots, x_m) \in \mathbb{Z}^m$ such that

$$\sum_{i=1}^{m} x_i j_i \equiv s - r \bmod N.$$

And a shortest path $\mathbf{x}$ is a path with minimum ℓ_1-norm. In contrast, a path from r to s in the directed circulant graph $DC_N(j_1, j_2, \ldots, j_m)$ can be described by a integer positive vector $\mathbf{x} = (x_1, \ldots, x_m) \in \mathbb{N}^m$ verifying the above equation, and a shortest path $\mathbf{x}$ is a path with minimum ℓ_1-norm.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 777–786, 2005.

For general graphs, finding a shortest path between two vertices is a well known and important problem. Efficient polynomial time algorithms have been developed for various routing problems. However, for the family of circulant graphs, there is an important distinction to be made, and that concerns the natural input size to a problem. For an arbitrary graph it is common to consider the input size to be N^2, which is the number of bits in its adjacency matrix. However, any circulant graph can be described by only m integers. In this representation the input size is $O(m \log N)$. Thus polynomial time algorithms for general graphs may exhibit exponential complexity in the special case of circulant graphs, for this compact input representation.

In [4] the authors establish relations between several routing applications for undirected circulant graphs and the problem of finding the shortest vector in the ℓ_1-norm in the above lattice. They present an algorithm which solves the Shortest-Loop problem in polynomial time for this input measure. In contrast, they show that the Shortest-Path problem is NP-hard for this concise representation.

The particular case $m = 2$, that is, undirected circulant graphs of degree four or distributed double-loop networks and directed circulant graphs of degree two or double-loop networks has been extensively studied, see the surveys [2, 9]). When N is given as a unary input and the time complexity is measured in terms of N, there are several shortest path algorithms for circulant graphs of degree four and for directed circulant graph of degree two, see for instance [6, 8, 10, 17, 18]. Typically, they require $O(N)$ arithmetic steps or $O(\log N)$ time for preprocessing and constant processing time at each node on the route. But a lower bound of the diameter for circulant graph and directed circulant graphs is $\Omega(\sqrt{N})$ (see [2]). So, they are in both cases exponential in the input size $\log N$. The paper [5] shows an algorithm to compute a shortest path in the circulant graph $C_N(1, h)$ (the so called chordal ring graphs) in $O(h/g + \log h)$ time, where $g = \gcd(h, N)$. Obviously, it has also exponential time complexity.

We remark in Section 2 that given a path $\mathbf{c}$, not necessarily a shortest one, in an undirected circulant graph then the problem of finding a shortest path is equivalent to problem of finding a vector that is closest to vector $-\mathbf{c}$ in the lattice defined by Equation 1 with respect to ℓ_1 norm. The well known paper [12] presents an algorithm for solving the Closest Vector Problem in a lattice given by a basis with respect to ℓ_1, ℓ_2 and ℓ_∞ norms. Moreover, fixed the dimension of the lattice, Kannan's algorithm is polynomial in the bit-size of the lattice's basis. So, fixed the number of the jumps m, the paper [12] provides a polynomial time algorithm of finding a shortest path in undirected circulant graphs. According with the above paragraph, this simple observation could be considered as a minor contribution of the present paper.

In this article we give a polynomial time deterministic algorithm to compute a shortest path between two vertices in any weighted circulant graph with N vertices and two jumps. Our algorithm only requires $O(\log N)$ arithmetic steps and the total bit operations is $O(\log^3 N)$. It is based on Closest Vector Problems for ℓ_1-norm. The paper is divided into five sections. In Section 2 we introduce some

known lattice concepts and show the relation between Shortest Path Problem and the Closest Vector Problem in a lattice for ℓ_1-norm. The Section 3 is devoted to describe a cubic polynomial time algorithm for solving the two dimensional CVP for ℓ_1-norm. As consequence of the two previous sections we obtain an algorithm for finding a shortest path in any undirected circulant graph of degree four. Section 4 is dedicated for finding a shortest path in double-loop networks. Finally, in Section 5, we analyze the problem in weighted circulant graphs.

2 Closest Vector Problem Versus Shortest Path Problem

The fundamental objects we are dealing with in this section are *lattices*, defined as a discrete subgroup of the space $\mathbb{R}^m$. Equivalently, a lattice is the set of integer linear combinations of some linearly independent vectors. Here we collect some definitions and well-known facts about lattices which can be found, for instance, in [7, 14, 16].

Let $\{\mathbf{b}_1, \ldots, \mathbf{b}_s\}$ be a set of linearly independent vectors in $\mathbb{R}^m$. The set

$$\mathcal{L} = \{\mathbf{z} \ : \ \mathbf{z} = c_1\mathbf{b}_1 + \ldots + c_s\mathbf{b}_s, \quad c_1, \ldots, c_s \in \mathbb{Z}\}$$

is called an *s-dimensional lattice* with *basis* $\{\mathbf{b}_1, \ldots, \mathbf{b}_s\}$.

One basic lattice problem is the *Shortest Vector Problem (SVP)*: given a lattice $\mathcal{L}$ and norm $\|\cdot\|$, finding a nonzero lattice with the smallest norm among all non-zero vectors in the lattice. Unfortunately, there are several indications that this problem is NP-complete. This study has suggested several definitions of a *reduced* basis for a lattice. The following concept is the generalization of the reduced basis concept in celebrated LLL algorithm [13] for lattices of rank 2 to an arbitrary norm [11].

Definition 1. *A basis* $\{\mathbf{u}, \mathbf{v}\}$ *is called reduced or Gauss-reduced respect to a norm* $\|\cdot\|$ *if* $\quad \|\mathbf{u}\|, \|\mathbf{v}\| \leq \|\mathbf{u}+\mathbf{v}\|, \|\mathbf{u}-\mathbf{v}\|$.

The algorithm in [11] computes a Gauss-reduced basis from a basis $\{\mathbf{u}, \mathbf{v}\}$ of the lattice $\mathcal{L} \subset \mathbb{Z}^2$ for any computable norm in $O(\log M)$ arithmetic steps where $M = \max(\|\mathbf{u}\|, \|\mathbf{v}\|)$ and a bound for total bit complexity is $O(\log^3 M)$. Notice that, possibly, swapping $\mathbf{u}$ and $\mathbf{v}$, we can always assume that $\|\mathbf{u}\| \leq \|\mathbf{v}\|$. Then, $\|\mathbf{u}\|$ and $\|\mathbf{v}\|$ are the two successive Minkowski minima.

Another related problem for which no polynomial time solution exists is the *Closest Vector Problem, CVP*:

Definition 2. *Given a basis B generating the lattice* $\mathcal{L} \subseteq \mathbb{R}^m$*, a vector* $\mathbf{v} \in \mathbb{R}^m$*, and a norm in* $\mathbb{R}^m$*; the Closest Vector Problem consists on finding a vector in the set* $\mathbf{v} + L$ *with minimum norm.*

The vertex-symmetry of circulants allows their analysis starting from any vertex, which simplifies their study. We may assume the routing is from vertex 0 to vertex $j \in \mathbb{Z}_N$. Using the well known Extended Euclidean Algorithm we compute a path $\mathbf{c} = (c_1, \ldots, c_m)$ from 0 to vertex j, that is, a solution of the congruence equation: $j_1x_1 + j_2x_2 + \cdots + j_mx_m \equiv j \bmod N$.

We consider the lattice $\mathcal{L}$ given in Equation (1), then we have the following observation:

Lemma 1. *With the above notation, we have:*
- A vector $\mathbf{w}$ *is a shortest path between* 0 *and* j *in* $C_N(j_1,\ldots,j_m)$ *if and only if* $\mathbf{w}$ *solves the CVP in the lattice* $\mathcal{L}$ *with norm* ℓ_1 *and for the vector* $\mathbf{c}$.
- A vector $\mathbf{w}=(x_1,\ldots,x_m)$ *is a shortest path between* 0 *and* j *in* $DC_N(j_1,\ldots,j_m)$ *if and only if* $\mathbf{w}$ *solves the CVP in the lattice* $\mathcal{L}$ *with norm* ℓ_1 *and for the vector* $\mathbf{c}$ *verifying that* $x_i \geq c_i, \quad i=1,\ldots,m.$

It is well known that CVP is NP-hard. However, for any fixed dimension CVP can be solved exactly in polynomial time for the Euclidean norm ℓ_2. The algorithm presented in [12] can also be adapted to find the ℓ_1 closest and the ℓ_∞ closest vectors. His algorithm requires cubic polynomial time (polynomial in the bit-size of the lattice's basis) for solving the two dimensional case.

In the next section we present an elementary cubic polynomial time algorithm for solving the two dimensional CVP with respect ℓ_1-norm which can also be extended for solving the shortest path in directed circulant graphs.

3 An Algorithm for Solving Two Dimensional CVP

The main problem addressed in this section is how to compute a vector that is closest to another given vector in a two dimensional lattice with respect to ℓ_1-norm.

For the rest of the paper we only consider the ℓ_1 norm. We denote by $\|\cdot\|$ this norm acting over a vector. As we are dealing with vectors $\mathbf{u}\in\mathbb{R}^2$, we denote their components by $\mathbf{u}=(u_1,u_2)$.

3.1 Reduction by a Vector

Given $\mathbf{u},\mathbf{v}$ in $\mathbb{R}^2$, with $\mathbf{v}\neq 0$, we can find $\alpha\in\mathbb{Z}$ such that the value $\|\mathbf{u}-\alpha\mathbf{v}\|$ is minimal, that is, we want to make $\mathbf{u}$ as short as possible by subtracting an integer multiple of $\mathbf{v}$ (see [11, 16]).

The main goal of this subsection is to obtain a such smallest vector with extra properties for our purpose. The algorithm `REDUCE` is an important tool of this paper:

Algorithm 1.
INPUT: $\mathbf{u},\mathbf{v}\in\mathbb{R}^2,\ \mathbf{v}\neq\mathbf{0}$.
OUTPUT: $\text{Reduce}_{\mathbf{v}}(\mathbf{u})\in\mathbf{u}+\mathbb{Z}\mathbf{v}\ /\ \|\text{Reduce}_{\mathbf{v}}(\mathbf{u})\| = \min\{\|\mathbf{u}+\alpha\mathbf{v}\|\ /\ \alpha\in\mathbb{Z}\}$.

- Select $i\in\{1,2\},\ j\in\{1,2\}\backslash\{i\}$ such that $|v_i|>|v_j|$. If $|v_1|=|v_2|$, then $i:=1$.
- Return the vector with minimum norm between:

$$\mathbf{u}-\left\lfloor\frac{u_i}{v_i}\right\rfloor\mathbf{v} \quad\wedge\quad \mathbf{u}-\left\lceil\frac{u_i}{v_i}\right\rceil\mathbf{v}.$$

If both have the same norm, return the one with ith non-negative component.

The next result collects several properties of this concept for later use.

Lemma 2. *Let* $\mathbf{u}, \mathbf{v} \in \mathbb{R}^2$ *be two vectors such that* $\mathbf{v} \neq 0$. *Let* $i \in \{1,2\}$, $j \in \{1,2\}\backslash\{i\}$ *as in Algorithm 1, that is,* $|v_i| > |v_j| \ \vee (i = 1 \ \wedge \ |v_1| = |v_2|)$. *We have the following properties:*

1. $\mathbf{r} = \text{Reduce}_{\mathbf{v}}(\mathbf{u}) \Rightarrow |r_i| < |v_i|$.
2. $\mathbf{h} \in \mathbf{u} + \mathbb{Z}\mathbf{v} \Rightarrow \text{Reduce}_{\mathbf{v}}(\mathbf{h}) = \text{Reduce}_{\mathbf{v}}(\mathbf{u})$.
3. $\text{Reduce}_{\mathbf{v}}(\text{Reduce}_{\mathbf{v}}(\mathbf{u})) = \text{Reduce}_{\mathbf{v}}(\mathbf{u})$.
4. $\text{Reduce}_{\mathbf{v}}(\mathbf{u}) = \text{Reduce}_{-\mathbf{v}}(\mathbf{u})$.

The properties 2, 3 and 4 in last Lemma show that $\text{Reduce}_{\mathbf{v}}(\mathbf{u})$ is invariant in the set $\mathbf{u} + \mathbb{Z}\mathbf{v}$.

In order to gauge the norm reduction performed by `REDUCE` procedure, the next result provides this bound.

Proposition 1. *Let* $\mathbf{u}, \mathbf{v} \in \mathbb{R}^2$ *be two vectors such that* $\mathbf{v} \neq \mathbf{0}$. *Let* $i \in \{1,2\}$, $j \in \{1,2\}\backslash\{i\}$ *as in Algorithm 1 with* $|v_i| = \beta |v_j|$ *for some* $1 \leq \beta \in \mathbb{R}$. *If* $|v_i| \leq |u_i|$ *and* $|u_j| \neq 0$, *then:* $\|\text{Reduce}_{\mathbf{v}}(\mathbf{u})\| \leq \frac{\alpha\left(1+\frac{1}{\beta}\right)+2}{2\alpha+2}\|\mathbf{u}\|$, *where* $|u_i| = \alpha |u_j|$, *for some* $\alpha \in \mathbb{R}$.

A first consequence of the above result is that: $\frac{\alpha\left(1+\frac{1}{\beta}\right)+2}{2+2\alpha} \leq 1$, because $\beta \geq 1$. And if $\beta > 1$ then the inequality is strict. This result can be extended for the cases $v_j u_j = 0$.

3.2 The Method's Core

We start with three vectors in $\mathbb{R}^2$, two of them linearly independent: $\mathbf{w} + \mathbb{Z} < \mathbf{u}, \mathbf{v} >$, $\text{rank}(\mathbf{u}, \mathbf{v}) = 2$.

We are going to find the shortest element in that set with respect to ℓ_1 norm. The method consists on recursively apply `REDUCE` algorithm to the "translation" vector $\mathbf{w}$ by some vectors in $\mathbb{Z} < \mathbf{u}, \mathbf{v} >$. Our purpose is to guarantee that each step reduces the vector's norm by a constant factor, until we reach some property. To perform this goal, we select a particular basis of the lattice $\mathbb{Z} < \mathbf{u}, \mathbf{v} >$:

Definition 3. *A lattice basis* $\{\mathbf{u}, \mathbf{v}\}$ *is call extra-reduced when:*

$$\text{Reduce}_{\mathbf{v}}(\mathbf{u}) = \mathbf{u} \ \wedge \text{Reduce}_{\mathbf{u}}(\mathbf{v}) = \mathbf{v}.$$

In order to study some properties of this kind of lattice basis, we classify vectors $\mathbf{u} = (u_1, u_2) \in \mathbb{R}^2$ in two types: **horizontal**, if $|u_1| \geq |u_2|$ and **vertical**, if $|u_1| < |u_2|$.

Lemma 3. *Let* $\{\mathbf{u}, \mathbf{v}\}$ *be two linear independent vectors: We have*

1. *If* $\{\mathbf{u}, \mathbf{v}\}$ *is a Gauss-reduced basis then one of the basis vectors is vertical and the other one is horizontal or both basis vectors are horizontal.*

2. *If* $\{\mathbf{u}, \mathbf{v}\}$ *is an extra-reduced basis then it is also Gauss-reduced and one of the vector is vertical and the other oner horizontal.*

Kaib and Schonrr showed in [11] how to get a reduced basis (referred to any norm, in particular ℓ_1) of lattice in two dimensions. Thanks Lemma 3, it is easy to reach an extra-reduced basis, by just applying `REDUCE` procedure.

Algorithm 2.
INPUT: $\mathbf{u}, \mathbf{v} \in \mathbb{R}^2$, Gauss-reduced basis of a lattice.
OUTPUT: $\mathbf{U}, \mathbf{V} \in \mathbb{R}^2$, extra-reduced basis of the same lattice.

1. Set the vectors $\mathbf{U}$, $\mathbf{V}$ so that $\{\mathbf{U}, \mathbf{V}\} = \{\mathbf{u}, \mathbf{v}\}$ and $\|\mathbf{U}\| \leq \|\mathbf{V}\|$.
2. **if** $\|\mathbf{U}\| < \|\mathbf{V}\|$, **do** $\mathbf{V} := \text{Reduce}_{\mathbf{U}}(\mathbf{V})$.
3. **else**
   1. **if** $\mathbf{U}$ and $\mathbf{V}$ are horizontal,
      i. **if** $|U_1| \neq |U_2|$, swap $\mathbf{U}$ and $\mathbf{V}$.
      ii. $\mathbf{V} := \text{Reduce}_{\mathbf{U}}(\mathbf{V})$.
   2. **else**
      i. **if** $\mathbf{U}$ is vertical, swap $\mathbf{U}$ and $\mathbf{V}$.
   3. **if** $U_2 < 0$, $\mathbf{U} := -\mathbf{U}$.
   4. **if** $V_1 < 0$, $\mathbf{V} := -\mathbf{V}$.

We will use then this extra-reduced basis to perform iterative reductions of the vector $\mathbf{w}$.

Algorithm 3.
INPUT: $\mathbf{w}, \mathbf{u}, \mathbf{v} \in \mathbb{R}^2$; $\{\mathbf{u}, \mathbf{v}\}$, extra − reduced basis $(\mathbf{u}, \text{hor.}, \mathbf{v}, \text{ver.})$.
OUTPUT: $\mathbf{W} \in \mathbf{w} + \mathbb{Z} < \mathbf{u}, \mathbf{v} >$, $|W_1| < |u_1|$, $|W_2| < |v_2|$.

- $\mathbf{W} := \mathbf{w}$.
- **while** $|W_1| \geq |u_1| \ \vee \ |W_2| \geq |v_2|$
  - **if** $|W_1| \geq |u_1|$, **do** $\mathbf{W} := \text{Reduce}_{\mathbf{u}}(\mathbf{W})$.
  - **else do** $\mathbf{W} := \text{Reduce}_{\mathbf{v}}(\mathbf{W})$.

From Lemma 2 and Proposition 1 we can obtain the following:

Lemma 4. *Algorithm 3 is correct and the number of performed loops is* $O(\log \|\mathbf{w}\|)$.

3.3 The Whole Process

We have reached a vector in $\mathbf{w} + \mathbb{Z} < \mathbf{u}, \mathbf{v} >$ with some properties. We need to conclude our job by getting the shortest vector among all. We will use the following technical result:

Lemma 5. *Let* $\{\mathbf{u}_1, \mathbf{u}_2\}$ *be a reduced basis (respect to any norm) of a lattice in* $\mathbb{R}^m$. *Let* $\mathbf{w} = \alpha_1\mathbf{u}_1 + \alpha_2\mathbf{u}_2$ *be a lattice vector* $(\alpha_1, \alpha_2 \in \mathbb{Z})$. *Then, we have:*

$$\|\alpha_1\mathbf{u}_1\|, \|\alpha_2\mathbf{u}_2\| \leq 2\|\mathbf{w}\|.$$

Let us suppose now we have reached a description $\mathbf{w} + \mathbb{Z} < \mathbf{u}, \mathbf{v} >$ of the original set, where $\{\mathbf{u}, \mathbf{v}\}$ is an extra-reduced basis ($\mathbf{u}$ is horizontal and $\mathbf{v}$ is vertical), and with $|w_1| < |u_1|,\ |w_2| < |v_2|$. Let $\mathbf{W}$ be a closest vector to $\mathbf{w}$ with respect ℓ_1 norm.

Then, $\mathbf{d} := \mathbf{W} - \mathbf{w} \in \mathbb{Z} < \mathbf{u}, \mathbf{v} >$ and $\|\mathbf{d}\| \leq 2\|\mathbf{w}\| = 2(|w_1| + |w_2|) < 2(|u_1| + |v_2|)$. We try to bound the coefficients of $\mathbf{d}$ as a lattice member: $\mathbf{d} = \alpha\mathbf{u} + \beta\mathbf{v}$. We distinguish two cases and applying previous Lemma 5

- $|u_1| \geq |v_2|$ then $\|\alpha\mathbf{u}\| \leq 2\|\mathbf{d}\| \Rightarrow |\alpha| \leq \frac{4(|u_1|+|v_2|)}{|u_1|+|u_2|} \leq 8$.
- $|u_1| < |v_2|$ then $\|\beta\mathbf{v}\| \leq 2\|\mathbf{d}\| \Rightarrow |\beta| \leq \frac{4(|u_1|+|v_2|)}{|v_1|+|v_2|} \leq 8$.

To sum up, jointing Algorithms 2, 3 and 4, we reach our goal:

Algorithm 4.

INPUT: $\mathbf{u}, \mathbf{v}$, extra-reduced basis ($\mathbf{u}$, hor. $\mathbf{v}$, ver.)
$\mathbf{w}$, with $|w_1| < |u_1|,\ |w_2| < |v_2|$
OUTPUT: $\mathbf{W}$, shortest vector in $\mathbf{w} + Z < \mathbf{u}, \mathbf{v} >$.

- $\mathbf{U} := \mathbf{u},\ \mathbf{V} := \mathbf{v}$.
- **if** $|U_1| < |V_2|$, swap $\mathbf{U}$ and $\mathbf{V}$.
- **for** $\alpha = [-8, \ldots, 8]$ **do**
  - $\mathbf{W}_\alpha := \text{Reduce}_{\mathbf{V}}(\mathbf{w} + \alpha\mathbf{U})$.
- Return a vector with minimum norm in $\{\mathbf{W}_\alpha \ / \ |\alpha| \leq 8\}$.

When studying lattices from a complexity point of view, it is customary to assume that the basis vectors (and therefore any lattice vector) have all rational coordinates. It is easy to see that rational lattices can be converted to integer lattices (i.e., sublattices of $\mathbb{Z}^m$) by multiplying all coordinates by an appropriate integer scaling factor.

If a, b are two integers, such that $b \neq 0$, we denote by $\text{quo}(a, b),\ \text{rem}(a, b)$ the unique integers verifying: $a = b \cdot \text{quo}(a, b) + \text{rem}(a, b),\ 0 \leq \text{rem}(a, b) < |b|$, i.e., $\text{quo}(a, b),\ \text{rem}(a, b)$ are the quotient and the remainder of the Euclidean division of a by b.

For every real number $x \in \mathbb{R}$, as usual we denote by $\text{sgn}\,(x)$ its sign.

In the case of lattices with integer coefficients, Algorithm 1 admits the following form:

Algorithm 5.

INPUT: $\mathbf{u}, \mathbf{v} \in \mathbb{Z}^2,\ \mathbf{v} \neq \mathbf{0}$.
OUTPUT: $\text{Reduce}_{\mathbf{v}}(\mathbf{u}) \in \mathbf{u} + \mathbb{Z}\mathbf{v} \ / \ \|\text{Reduce}_{\mathbf{v}}(\mathbf{u})\|$
$= \min\{\|\mathbf{u} + \alpha\mathbf{v}\| \ / \ \alpha \in \mathbb{Z}\}$.

- Find $i \in \{1, 2\},\ j \in \{1, 2\} \backslash \{i\}$ such that $|v_i| > |v_j|$. If $|v_1| = |v_2|$, select $i := 1$.
- Return the vector with minimum norm between:

$$\mathbf{u} - \text{quo}(u_i, v_i)\mathbf{v},\ \mathbf{u} - (\text{quo}(u_i, v_i) + \text{sgn}\,(v_i))\mathbf{v}.$$

If both share the same norm, return $\mathbf{u} - \text{quo}(u_i, v_i)\mathbf{v}$.

It is straightforward to check both Algorithm 1 and Algorithm 5 have same output for integer vectors. Then, clearly given an undirected circulant graph $C_N(j_1, j_2)$ and vertex $j \in \mathbb{Z}_N$, we can decide if there exists a shortest path from vertex 0 to vertex j and, in the affirmative case, we can compute one on $O(\log^3 N)$ bit operations.

4 Directed Circulant Graphs

Our method can be easily extended to *directed circulant graphs* $DC_N(j_1, j_2)$. First, we need to introduce the concept of *positive reduction of a vector*. Given two vectors $\mathbf{u} = (u_1, u_2)$ and $\mathbf{v} = (v_1, v_2) \neq \mathbf{0}$, we are looking for $\alpha \in \mathbb{Z}$ such that the value $\|\mathbf{u} - \alpha\mathbf{v}\|$ is minimal and having both components positive. In general, $\mathbf{u} - \alpha\mathbf{v}$ with both components positive may not achieve the minimum norm over all integer component $\mathbf{u} - \beta\mathbf{v}$, with α, β integral.

Algorithm 6.

INPUT $\mathbf{u}, \mathbf{v} \in \mathbb{Z}^2,\ \mathbf{v} \neq 0$

OUTPUT $\mathrm{PRed}_{\mathbf{v}}(\mathbf{u}) \in (\mathbf{u} + \mathbb{Z}\mathbf{v}) \cap \mathbb{N}^2$,
$\|\mathrm{PRed}_{\mathbf{v}}(\mathbf{u})\| \leq \|\mathbf{u} + \alpha\mathbf{v}\|\ \forall \alpha \in \mathbb{Z} / \mathbf{u} + \alpha\mathbf{v} \in \mathbb{N}^2 \quad \vee$
$\emptyset$, if $(\mathbf{u} + \mathbb{Z}\mathbf{v}) \cap \mathbb{N}^2 = \emptyset$

1. Find $i \in \{1, 2\},\ j \in \{1, 2\} \backslash \{i\}$ such that $|v_i| > |v_j|$. If $|v_1| = |v_2|$, select $i := 1$.
2. Set $\varepsilon = \begin{cases} -1, & \text{if } |v_1| \geq |v_2| \\ 1\ , & \text{if } |v_1| < |v_2|. \end{cases}$
3. Compute $\Delta := \varepsilon\ \mathrm{sgn}\,(u_1 v_2 - u_2 v_1)$.
4. If $v_j = 0$,
$$B := \Delta \mathrm{sgn}\,(v_i)\,.$$
   1. If $B = -1$, Output $\emptyset$.
   2. If $B \geq 0$, Output $\mathbf{u} - \mathrm{sgn}\,(v_i)\,\mathrm{quo}(u_i, |v_i|)\mathbf{v}$.
5. If $v_j \neq 0$,
$$A := -\Delta \mathrm{sgn}\,(v_j)\,,\ B := \Delta \mathrm{sgn}\,(v_i)\,.$$
   1. If $(A = -1 \wedge B = -1)$, Output $\emptyset$.
   2. If $(A = -1 \wedge B \geq 0)$, Output $\mathbf{u} - \mathrm{sgn}\,(v_i)\,\mathrm{quo}(u_i, |v_i|)\mathbf{v}$.
   3. If $(A \geq 0 \wedge B = -1)$, Output $\mathbf{u} - \mathrm{sgn}\,(v_j)\,\mathrm{quo}(u_j, |v_j|)\mathbf{v}$.
   4. If $(A \geq 0 \wedge B \geq 0)$,
      i. $\mathbf{w} := \mathbf{u} - \mathrm{sgn}\,(v_i)\,\mathrm{quo}(u_i, |v_i|)\mathbf{v}$.
      ii. If $\mathbf{w} \in \mathbb{N}^2$, Output $\mathbf{w}$.
      iii. Else, Output $\emptyset$.

It is easy to check that the previous algorithm is correct and a bound for the bit complexity on computing $\mathrm{PRed}_{\mathbf{v}}(\mathbf{u})$ is $O(\log^2 M)$, where $M = \max(\|\mathbf{u}\|, \|\mathbf{v}\|)$.

Once this tool is fixed, let us describe the method to reach a shortest path in a directed circulant graph. Firstly, we act as seen in the previous section to compute an extra reduced basis of the associated lattice $\{\mathbf{u}, \mathbf{v}\}$, and a shortest path for the corresponding undirected circulant graph $\mathbf{w}$.

We can state that there always exists one path in the directed circulant graph with bounded length.

Lemma 6. *Let* $\mathbf{w}, \mathbf{u}, \mathbf{v} \in \mathbb{Z}^2$*, such that* $\{\mathbf{u}, \mathbf{v}\}$ *is an extra reduced basis for the lattice they generate. Then,* $\exists \mathbf{d} \in (\mathbf{w} + \mathbb{Z} < \mathbf{u}, \mathbf{v} >) \cap \mathbb{N}^2,\ \|\mathbf{d}\| \leq 6 \max\{\|\mathbf{u}\|, \|\mathbf{v}\|\}$.

Proof. Let $M = \max\{\|\mathbf{u}\|, \|\mathbf{v}\|\}$. We consider the translated lattice

$$\mathbf{w} - (2M, 2M) + \mathbb{Z} < \mathbf{u}, \mathbf{v} > .$$

By Algorithm 3, this set contains an element $\mathbf{z}$, with $|z_1| < |u_1|,\ |z_2| < |v_2|$. So, $\|\mathbf{z}\| \leq 2M$. Clearly $\mathbf{z} + (2M, 2M)$ belongs to the set $(\mathbf{w} + \mathbb{Z} < \mathbf{u}, \mathbf{v} >) \cap \mathbb{N}^2$, and its norm is bounded by $6M$. □

Finally, we follow a similar argument than in Section 3 to reach the shortest path for the directed graph.

Algorithm 7.

INPUT: $\mathbf{w} \in \mathbb{Z}^2$, $\{\mathbf{u}, \mathbf{v}\}$, extra reduced basis.

OUTPUT: $\mathbf{d}$, shortest element in $(\mathbf{w} + \mathbb{Z} < \mathbf{u}, \mathbf{v} >) \cap \mathbb{N}^2$.

- Find a shortest element $\mathbf{z}$ in $\mathbf{w} + \mathbb{Z} < \mathbf{u}, \mathbf{v} >$.
- **if** $\|\mathbf{u}\| \geq \|\mathbf{v}\|$
  - **for** $\alpha = -16, \ldots, 16$ **do** $\mathbf{d}_\alpha := \mathrm{PRed}_{\mathbf{v}}(\mathbf{z} + \alpha\mathbf{u})$.
- **else**
  - **for** $\alpha = -16, \ldots, 16$ **do** $\mathbf{d}_\alpha := \mathrm{PRed}_{\mathbf{u}}(\mathbf{z} + \alpha\mathbf{v})$.
- Return a vector with minimum norm in $\{\mathbf{d}_\alpha \ / \ |\alpha| \leq 16\}$.

Proof. Let $\mathbf{w}, \mathbf{u}, \mathbf{v} \in \mathbb{Z}^2$, such that $\{\mathbf{u}, \mathbf{v}\}$ is an extra reduced basis for the lattice they generate. Let $\mathbf{W}$ be a shortest element in a translated lattice $\mathbf{w} + \mathbb{Z} < \mathbf{u}, \mathbf{v} >$ and let $\mathbf{d}$ be a shortest element in $(\mathbf{w} + \mathbb{Z} < \mathbf{u}, \mathbf{v} >) \cap \mathbb{N}^2$. We have: $\mathbf{d} - \mathbf{W} = \alpha\mathbf{u} + \beta\mathbf{v}$, verifying $|\alpha| \leq 16$ if $\|\mathbf{u}\| \geq \|\mathbf{v}\|$, $|\beta| \leq 16$ if $\|\mathbf{v}\| \geq \|\mathbf{u}\|$.

Let $M = \max\{\|\mathbf{u}\|, \|\mathbf{v}\|\}$, by Lemma 5 and Lemma 6 and since $\{\mathbf{u}, \mathbf{v}\}$ is an extra reduced basis, we have:

$$\|\alpha\mathbf{u}\|, \|\beta\mathbf{v}\| \leq 2\|\mathbf{d} - \mathbf{w}\| \leq 2\|\mathbf{d}\| + 2\|\mathbf{W}\| \leq 12M + 4M \leq 16M.$$

□

5 Weighted Circulant Graphs

In this section, we consider weighted circulant graphs with two jumps $C_N(j_1, j_2)$ and weights $w = (w_1, w_2)$.

Theorem 8. *Given a circulant graph* $C_N(j_1, j_2)$ *with weights* $w = (w_1, w_2)$ *we can find a shortest path on cubic polynomial time.*

Proof. The distance of a path $\mathbf{c} = (c_1, c_2)$ in the weighted circulant graph is

$$\|\mathbf{c}\|_w = w_1|c_1| + w_2|c_2|.$$

Let $\mathbf{c} \in \mathbb{Z}^2$ then $\|\mathbf{c}\|_w = \|\Phi(\mathbf{c})\|_{\ell_1}$ where Φ is the injective group homomorphism $\Phi : \mathbb{Z}^2 \to \mathbb{Z}^2,\quad \Phi((x, y)) = (w_1 x, w_2 y)$. Let $j \in \mathbb{Z}_N$ be a vertex of the graph and let $\mathbf{u}, \mathbf{v}$ be an extra-reduced basis of the circulant graph $C_N(j_1, j_2)$. By Section 3 we compute on cubic polynomial time a shortest path $\mathbf{c}$ from vertex 0 to vertex j. Let $\mathbf{d}$ a solution to CVP for the lattice generated by $< \Phi(\mathbf{u}), \Phi(\mathbf{v}) >$ and target vector $\Phi(\mathbf{c})$, then $\Phi^{-1}(\mathbf{d})$ is a shortest path in the weighted undirected circulant graph. □

The algorithm in the above theorem can be adapted in a natural way to weighted directed circulant graphs.

References

1. R. Beivide, E. Herrada, J.L. Balcázar and A. Arruabarrena. *Optimal Distance Networks of Low Degree for Parallel Computers.* IEEE Transactions on Computers, Vol. C-40, No. 10, pp. 1109-1124, 1991.
2. J.-C. Bermond, F. Comellas and D.F. Hsu. *Distributed Loop Computer Networks: A Survey.* Journal of Parallel and Distributed Computing, Vol. 24, pp. 2-10, 1995.
3. F.T. Boesch and R. Tindell. *Circulants and their connectivity.* J. Graph Theory, Vol. 8, pp. 487-499, 1984.
4. J.-Y. Cai, G. Havas, B. Mans, A. Nerurkar, J.-P. Seifert and I. Shparlinski. *On Routing in Circulant Graphs.* Proc. Fifth Annual International COCOON- 1999, LNCS vol. 1627, Springer-Verlag, T. Asano, H. Imai, D.T. Lee, S. Nakano, and T. Tokuyama (Eds.), pp. 360-369.
5. N. Chalamaiah and B. Ramamurthy. *Finding shortest paths in distributed loop networks.* Information Processing Letters, Vol. 67, pp. 157-161, (1998).
6. Y. Cheng and F. K. Hwang. *Diameters of Weighted Double Loop Networks*, Journal of Algorithms 9, 401-410, 1988.
7. M. Grötschel, L. Lovász and A. Schrijver, *Geometric algorithms and combinatorial optimization*, Springer-Verlag, Berlin, 1993.
8. D. J. Guan. *An Optimal Message Routing Algorithm for Double-Loop Networks.* Information Processing Letters 65(5): 255-260, 1998.
9. F. K. Hwang. *A complementary survey on double-loop networks*, Theoretical Computer Science 263, 2001. 211-229.
10. F. K. Hwang. *A survey on multi-loop networks*, Theoretical Computer Science 299, 2003. 107-121.
11. M. Kaib and C.P. Schnorr: "The Generalized Gauss Reduction Algorithm". Journal of Algorithms **21**, 3 (1996): 565-578.
12. R. Kannan. *Minkoswski's convex body theorem and integer programing*, *Mathematics of operation research* , **12**(3), 415–440, 1987.
13. A. K. Lenstra, H. W. Lenstra and L. Lovász. 'Factoring polynomials with rational coefficients', *Mathematische Annalen*, **261**, 515–534, 1982.
14. L. Lovász and H. Scarf: "The Generalized Basis Reduction Algorithm". Mathematics of Operations Research **17**, 3 (1992): 751-764.
15. B. Mans. *Optimal Distributed algorithms in unlabeled tori and chordal rings* Journal of Parallel and Distributed Computing, Vol. 46, pp. 80-90, 1997.
16. D. Micciancio and S. Goldwasser. *Complexity of Lattices Problems*, The Kluwer International Series in Engineering and Computer Science, vol. 671, 2002.
17. K. Mukhopadhyaya and B.P. Sinha. *Fault-Tolerant Routing Algorithm in distributed Loop Networks.* IEEE Transactions on Computers 44(12): 1452-1456, 1995.
18. Yu-Liang Liu, Yue-Li Wang and D. J. Guan. *An Optimal Fault-Tolerant Routing Algorithm for Double-Loop Networks.* IEEE Transactions on Computers 50(5): 500-505, 2001.
19. J. Žerovnik, and T. Pisanski. *Computing the Diameter in Multiple-Loop Networks.* J. Algorithms 14(2): 226-243, 1993.

A Linear Time Algorithm for Finding a Maximal Planar Subgraph Based on PC-Trees

Wen-Lian Hsu

Institute of Information Science, Academia Sinica, Taipei, Taiwan

Abstract. In Shih & Hsu, a planarity test was introduced utilizing a data structure called PC-trees, generalized from PQ-trees. They illustrated that a PC-tree is more natural in representing planar graphs. Their algorithm starts by constructing a depth-first-search tree and adds all back edges to a vertex one by one. An important feature in the S&H algorithm is that, at each iteration, at most two terminal nodes need to be computed and the unique tree path between these two nodes provides essentially the boundary path of the newly formed biconnected component.

In this paper we modify their PC-tree algorithm and introduce the deferred planarity test (DPT), which has the added benefit of finding a maximal planar subgraph (MPS) in linear time when the given graph is not planar. DPT is an incremental algorithm, which only computes a partial terminal path at each iteration. DPT continually deletes back edges that could create a violation to the formation of those partial terminal paths so that, at the end, the subgraph constructed is guaranteed to be planar.

The key to the efficiency of the S&H and the DPT algorithms lies in their management on the creation and destruction of biconnected components in which the PC-tree plays a major role. Previously, there have been reports that the MPS problem can be solved in linear time. However, there was no concrete data structure realizing them.

1 Introduction

Given an undirected graph, the planarity test is to determine whether there exists a clockwise edge ordering around each vertex, such that the graph can be drawn in the plane without any crossing edges. Linear time planarity test was first established by Hopcroft and Tarjan [5] based on a "*path addition* approach." A "*vertex addition* approach", originally developed by Lempel, Even and Cederbaum [13], was later improved by Booth and Lueker [1] (hereafter, referred to as B&L) to run in linear time using a data structure called a "PQ-tree". Several other approaches have also been developed for simplifying the planarity test (see for example [2], [3], [16], [19], [21]) and the embedding algorithm [14]. Shih and Hsu [18] (hereafter referred to as S&H) developed a very simple linear time test based on PC-trees. Further exposition of S&H algorithm can be found in Hsu and McConnell [9]. An earlier version [17] of Shih and Hsu [18] has been referred to as the simplest linear time planarity test by Thomas [20] in his lecture notes.

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 787–797, 2005.

Two attempts [10], [12] to use PQ-tree (following the LEC approach) in the construction of a maximal planar subgraph (MPS) algorithm of $O(n^2)$ failed as indicated by Junger, Leipert and Mutzel [11]. Currently, the fastest algorithm for the MPS problem takes $O(m \log n)$ time by Cai, Han and Tarjan [3]. Previously, there have been claims [6], [7] that the MPS problem can be solved in linear time. However, there was no concrete data structure realizing them.

In this paper we illustrate how to modify S&H algorithm to yield another planarity test, the deferred planarity test (DPT), which has the added benefit of finding a maximal planar subgraph in linear time when the given graph is not planar. Relative to the original PC-tree algorithm of S&H, which utilizes full back edge information at all times, DPT considers only existing back edges. The key to the efficiency of the DPT is its management on the creation and destruction of biconnected components in which the PC-tree plays a major role.

In Section 2 we review some background information in the S&H algorithm. PC-trees are discussed in Section 3. Important properties of S&H algorithm are reviewed in Section 4, which form the basis of the DPT algorithm. Section 5 describes the DPT algorithm. Complexity of the DPT algorithm is analyzed in Section 6.

2 Background

Let n be the number of vertices and m the number of edges of the graph G. Construct a depth-first search tree T for G. Note that every non-tree edge of G must be a back edge from a vertex to one of its ancestors. Let 1, ..., n be the order resulting from a postorder traversal of T. So the order of a child is always less than that of its parent. Denote the subtree of T with root i by T_i. Initially, we include all edges of T, namely the depth-first-search tree, in the embedding. Then, at iteration i, we add all back edges one by one from the descendants to node i and update the embedding. Let r be a child of i such that there is a node in T_r with a back edge to i. Then at the end of this iteration there is a biconnected component generated containing r. In fact, for each such child of i there is such a biconnected component generated around node i. Whenever a biconnected component is generated, we create its internal embedding, store it, and use a subset of vertices (called essential nodes) in its boundary cycle as representatives to be used for future embedding.

Denote the largest neighbor of a node i by $h(i)$. Sort the children of each node of T according to the ascending order of their labels. At each iteration i, we consider the embedding of the back edges from the descendants to i and revise the tree accordingly. Denote the revised tree at the end of iteration i by T^i.

Initially, there is no biconnected component. Consider the first iteration i such that a child subtree T_r of i (with root r) has a back edge to i. A biconnected component containing r will be generated at the end of this iteration. The major work involved in generating a biconnected component is to determine its boundary, its inner vertices (those that should be embedded inside) and its outer vertices. S&H used a labeling algorithm to annotate the above based on the following notations and properties:

Definition 1. *Classify the nodes in T_r into four types: A node is type 1 if it has no back edge; type 2 if it has a back edge to i and no other back edge; type 3 if it has a back edge to i and a back edge to a node $> i$; and type 4 if all of its back edges are to nodes $> i$.*

Note that a type 1 leaf in T_r can be deleted without affecting the planarity of the remaining graph. Such a deletion can be propagated recursively. Hence, we could assume there is no type 1 leaf in T_r. A back edge from a subtree of T_r to a node $> i$ is also referred to as a *remote back edge* at iteration i.

Definition 2. *A leaf in T_r is full if it is type 2, partial if it is type 3, and empty if it is type 4. A subtree T_v is*

1. *full if it contains only type 1 and type 2 nodes*
2. *partial if it either contains a type 3 node, or contains both a type 2 node and a type 4 node*
3. *empty if it contains only type 1 and type 4 nodes.*

A node v is full (respectively, partial, empty) if T_v is full (respectively, partial, empty). Define a terminal node in T_r to be a partial node whose children are either full or empty.

Assume the graph is planar. We have the following properties from [18]:

(a) *The parent of a partial node is partial.*
(b) *A partial node must contain a terminal node as a descendant.*
(c) *There are at most two terminal nodes in T_r.*
(d) *Any node v with two descendant terminal nodes satisfies $h(v) \leq i$.*

3 PC-Trees

To facilitate the representation of biconnected components, the notion of PC-trees was introduced in [18]: a tree is a PC-tree if its nodes can be divided into two types: P-nodes and C-nodes, where the neighbors of a P-node (denoted by a circle) can be permuted arbitrarily and the neighbors of a C-node (denoted by a double circle) must observe a cyclic order, which can only be reversed. S&H used a PC-tree to represent the embedding of a partial planar graph, in which a P-node denotes a regular vertex of the graph and a C-node denotes a biconnected component as illustrated in Figure 1.

Unlike the PQ-tree, a PC-tree is unrooted. An important property of the embedding of a partial planar graph at iteration i is that each vertex on the boundary of a biconnected component should have back edges to nodes greater than i; for otherwise, it can be contracted.

S&H showed that, when the graph is planar, there is a unique *terminal path* TP_i at each iteration i, which basically form the *representative boundary cycle* (RBC), iTP_ii, of the resultant biconnected component. The RBC will enclose all full subtrees in the inside, and leave empty subtrees on the outside. Figure 2

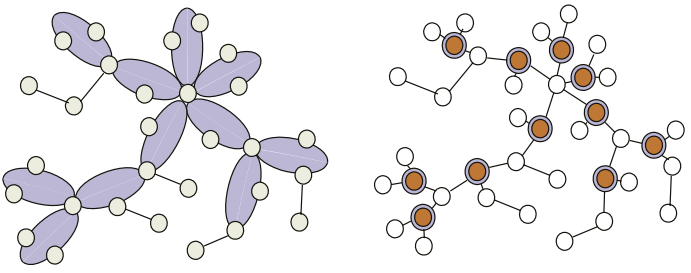


Fig. 1. An embedding of a planar graph and its corresponding PC-tree

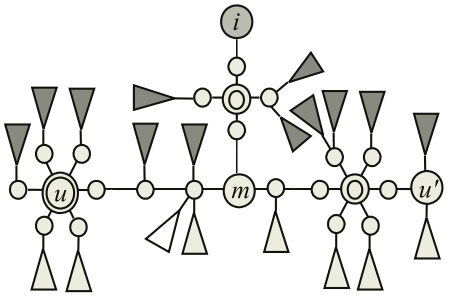


Fig. 2. The unique terminal path between two terminal nodes u and u'

illustrates a terminal path in the current PC-tree. Define node i to be the *head* of this RBC. Define the *essential nodes* of an RBC to be those that have a back edge to a node $> i$ or have at least an empty child. The essential nodes and the head are the only ones on the RBC relevant to future embedding and the remaining nodes are contracted. The RBC will be stored as a circular doubly linked list of the essential nodes and the head. To distinguish it from the original edges of the graph, we refer to the connections on the RBC as *links*. To maintain a tree-like structure for the current embedding, represent the biconnected component by a C-node, say, w, whose parent is i and whose children are the essential nodes in the terminal path. Define the two *end nodes* of w to be the two neighbors of i in its RBC; let $head(w) = i$ and $h(w) = i$.

Since old biconnected components can be merged and new ones are created, the parent pointers of the children of C-nodes can be changed frequently. To avoid this overhead, S&H adopted the strategy of B&L by borrowing parent pointers through the neighbors in its boundary cycle and keep parent pointers only for a few necessary nodes (to be explained later). Therefore, for most children of a C-node, their parent pointers are "virtual," which are there only to illustrate the algorithm conceptually.

If the graph is planar, we have the following property on full children of a partial C-node:

(e) *Let u_1 be a partial C-node of P_u. Then the set of full children plus head(u_1) are consecutive in its RBC. If u_1 is an internal node of P_u, let its child v and head(u_1) be u_1's two neighbors in P_u, then at least one neighbor of v in the RBC is full. If both neighbors of v in its RBC are full, then all children of u_1 not on P_u are full and every ancestor t of u_1 in P_u satisfies $h(t) \leq i$.*

The main operation in S&H is the use of a tree traversal algorithm to identify all full nodes and partial nodes in T_r. Whenever a node on the RBC of a C-node is traversed, S&H used a parallel search algorithm along the boundary links to locate its current parent.

Note that, in Lempel, Even and Cederbaum's planarity test [13], internal nodes of the PQ-trees are there only to keep track of feasible permutations. However, in the PC-tree approach, every P-node is an original node of the graph, every C-node represents a biconnected component in the partial embedding, and nodes adjacent to the incoming node can be scattered anywhere, both as internal nodes and as leaves in our PC-tree. Thus, a PC-tree is a more natural representation.

4 Important Properties Derived from the S&H Algorithm

Three properties of the S&H algorithm that form the basis for the DPT are described below.

Theorem 1. *A graph G is planar iff, when we perform the S&H algorithm, there are at most two terminal nodes and conditions (c), (d), and (e) are satisfied at every iteration.*

Another important property is related to the essential nodes of an RBC. Again, assume the given graph is planar. Consider an iteration i of the S&H algorithm that has a back edge from a node in a child subtree T_r to i. Let w denote the corresponding C-node formed at the end of this iteration. Consider the essential nodes of w. For each essential node v. Denote its empty children by $v_1, \ldots, v_k$. For each v_m, $m = 1, \ldots, k$, define

$i(v_m) = \min \{ t |$ (s, t) is a back edge from a node s in T_{vi} to $t\}$,

which is the smallest ancestor that a back edge from a node in T_{vm} goes to. Since any back edge from these children subtrees must go to a node larger than i, we have $i(v_m) > i$ for $m = 1, \ldots, k$. At the $i(v_m)$-th iteration, there is a back edge from a node, say, x, in T_{vm} to $i(v_m)$, which would initiate a traversal from x through v_m to $i(v_m)$ along the tree path for the first time. Such a traversal will leave a label $i(v_m)$ on v. Thus, each essential node v could receive several labels. Let $E(w)$ be the collection of labels, $t_1, \ldots, t_p$ (in the ascending order), received by the essential nodes of w. For each label t, let $S(t)$ be the set of essential nodes receiving the label t. We have the following consecutive property:

Theorem 2. *Labels $t_1, \ldots, t_p$(in the ascending order) satisfy that nodes in $[S(t_m) \cup S(t_{m+1}) \cup \ldots \cup S(t_p)]$, for each $m = 2, \ldots, p$, are consecutive in*

the RBC of w, say from $e_{m1}, e_{m2}, \ldots$ *to* e_{mk} *in the counter-clockwise order. Furthermore, no node in* $\{i\} \cup [S(t_1) \cup S(t_2) \cup \ldots \cup S(t_{m-1})]$ *can fall within the set* $\{ e_{m2}, \ldots, e_{m(k-1)}\}$.

In other word, Theorem 2 says that, for each m, $m = 2, \ldots, p$, if we use nodes in $\{i\} \cup [S(t_1) \cup S(t_2) \cup \ldots \cup S(t_{m-1})]$ as *delimiters* in the circular linked list for w, then all delimiters are consecutive in the list and there exist one pair of nodes, say $\{e', e''\}$, for which nodes in $[S(t_m) \cup S(t_{m+1}) \cup \ldots \cup S(t_p)]$ must be exactly those nodes in between e' and e'' in the list(see Figure 3). Note that e' or e'' could be the same as e_{m1} or e_{mk}.

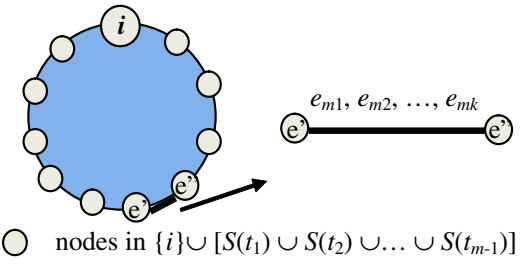


Fig. 3. An example illustrating Theorem 2

This theorem can be proved by virtue of the fact that any violation will produce a Kuratowski subgraph. Because DPT does not use information on remote back edges, no essential node can be readily identified at iteration i. They will emerge in later iterations.

Finally, we have the following fundamental theorem for the DPT algorithm.

Theorem 3. *In performing the S&H algorithm, if the set of back edges do not violate the properties described in Theorem 2, then there can be no more than 2 terminal nodes and conditions (c), (d), and (e) are satisfied at every iteration. Hence, the graph G must be planar.*

5 Deferred Planarity Test (DPT)

Although the DPT is designed to solve the MPS problem, our description will focus on how it is used for planarity test. Along the way, we will indicate the extra steps needed for the MPS problem.

Theorem 1 is the basis for the S&H algorithm. However, the conditions in Theorem 1 cannot be used directly for the MPS problem because, at iteration i, a back edge from a node in T_i to a node $> i$ could later be deleted, and that could affect the determination of terminal nodes at iteration i (namely, condition (b) does not hold for non-planar graphs). Therefore, any MPS algorithm should not make use of information on back edges to nodes $> i$ before the $(i+1)$-th iteration. Our idea is to modify the S&H algorithm so as to guarantee that no more than

two terminal nodes would be created at any iteration. The determination of terminal nodes in an iteration is deferred until all relevant back edges have been determined in later iterations. Hence, the modified algorithm is called the *deferred planarity test*. All terminal nodes must eventually be determined at the final iteration of the DPT.

The preprocessing stage of the DPT is the same as that in the S&H algorithm. Find a depth-first-search tree T and get a postorder for nodes in T. A PC-tree will be used to represent the partial graph at each iteration. The major operation of DPT is described in the next section.

5.1 Back Edge Traversal

The DPT adopts a labeling routine to embed the back edges, which also controls the creation and destruction of biconnected components during the embedding process. There are two kinds of traversals in the labeling routine: tree edge traversal on the PC-tree, and link traversal on a RBC. The former is the main routine that assigns labels to nodes and edges traversed, whereas the latter is only used to identify the parent of a node based on boundary link traversal. During the edge traversal, we could encounter a node whose parent is yet to be identified, then DPT initiates a link traversal on the RBC of the corresponding component to find that parent, and renew the edge traversal starting from this parent node.

Consider a traversal from a node s with back edge (s, i) upward to i. When a node v first re-ceives a label i, it indicates that node v is now involved in some component of i. However, there is no C-node for the component at this point yet. Node v still keeps its original parent in the PC-tree. When a node v receives a label i first and a label j next, let parent$(v) = i'$.(note that the original component of i containing v could have been merged into other components) Then node v was an essential node on the RBC of a component $B_{i'}$ of i', but v is now involved in some component B_j of j, and $B_{i'}$ will be merged into B_j. In general, if a node v receives more than 2 labels with i, j being the last two, let parent$(v) = i'$. Then node v was an essential node on the RBC of a component $B_{i'}$ of i', but v is now involved in some component B_j of j, and $B_{i'}$ will be merged into B_j.

We now describe the details of the labeling algorithm. To maintain consistency with previous notations we use indices i, j, k to indicate that there is a component of i first traversed through a back edge traversal from a neighbor of j (thus, this component is merged into a component of j), and such a component for j is first traversed through a back edge traversal from a neighbor of k.

Consider the operations at iteration k (and then refer back to iterations j and i whenever applicable). In S&H algorithm, the strategy of dividing nodes in T_r into full nodes, partial nodes and empty nodes at iteration k can no longer be applied here since DPT does not make use of the information on remote back edges. The DPT algorithm labels both nodes and edges. Each node has a *label stack* so that a node can receive many labels (denoted by $[\ldots,i,j,k]$) with the most recent one (i.e. k) on top. Each edge can receive a pair of ordered labels

denoted by $< i, j >$. These labels, once assigned, will remain throughout the algorithm.

At iteration k, we add the back edges from nodes in T_r to k one by one starting from the lowest-indexed neighbor of i. Since we do not use information on back edges to nodes $> k$, terminal nodes at the k-th iteration cannot be identified right away. Each back edge, say, (u,k), embedded to the graph could affect the identification of terminal nodes for components formed in previous iterations (e.g. i and j) . DPT checks whether the conditions in Theorem 3 are valid for all previous iterations. If not, then DPT removes those other back edges responsible for such violations. To describe such an effect, we will trace the path from u to k in the PC-tree. Our main back edge traversal algorithm is described below. Assign label $<k, \emptyset>$ to the back edge (u,k) for every neighbor u of k.

We use the following subroutines: Label(u), Find(u), and Eliminate(subtree). Roughly speaking, Label(u) assigns a label to the top of the stack for the node u; Find(u) returns the C-node whose RBC contains node u; Eliminate(subtree) deletes all remote back edges for the nodes in the subtree. Define the set *Delimiter* to keep track of all delimiters of a newly generated C-node.

The Back Edge Traversal Algorithm

1. Initialize the set, Delimiter, to be $\emptyset$. Let u be the first neighbor of k to be labeled through the edge (u,k).
2. While there is still a back edge (u,k) do Label(u).
   Let $u \leftarrow$ the next vertex in the neighbor list of k. If $u > j$, then generate the C-node, w, for the component of j containing i. Make all nodes in Delimiter children of w. *head*(w) $\leftarrow j$. Delimiter $\leftarrow \emptyset$.
   end while

Subroutine Label(u)

/* The detailed Eliminate(subtree) steps are not given here. They will appear in the full version of the paper.*/
Let u_1 be the predecessor of u in the traversal (/* Note that the traversal will trace out a tree path from a neighbor of k to u */).
if u is an unlabeled P-node **then**
  label it [k] (/* u is now included in a component of k*/). Label the tree edge (u, parent(u)) by $<k, \emptyset>$. Label(parent(u)).
end if
if u is a P-node with only one label i **then**
  if edge (u_1,u) has label $<k, \emptyset>$ **then**
    Delimiter $\leftarrow$ Delimiter $\cup \{u\}$. $u' \leftarrow u$.
  end if
  while u' has only one label i and $u' \neq i$ **do**
    give u' the label k on top of its label stack. Label the edge (u', parent(u')) by $<i, k>$.
    $u' \leftarrow$ parent(u')
  end while

if $u' = i$ and has no more than 1 label **then**
let Delimiter ← Delimiter ∪ $\{u\}$; Label(i).
else
(* u' has two labels [i, j] *) Label(u').
end if
end if
if u is a P-node with more than one label [...,i,j] with i, j being the last two **then**
if edge (u_1,u) has label $<k, \emptyset>$ **then**
Delimiter ← Delimiter ∪ $\{u\}$.
end if
if $j < k$ **then**
if parent(u) = $\emptyset$ **then**
w ← Find(u)
else
w ←parent(u)
end if
Eliminate(the connected component associated with nodes in the forbidden area)
Label(*head*(w)).
else
(* $j = k$ *) stop
end if
end if

Subroutine Find(u)
/* This procedure will return the C-node that contains u on its boundary cycle through *parallel search* */
if parent(u) ≠ $\emptyset$ **then**
Find(u) ← parent(u);
else
Search the neighbors of u in the doubly linked list in both directions until a node, say v, with a nonempty parent pointer is found.
end if
Find(u) ← v.

Subroutine Eliminate(subtree)
Deletes all back edges for nodes in the subtree.

5.2 *C*-Nodes in the DPT Algorithm

A C-node on a path is said to have its *correct* side determined if an essential node other than its two neighbors on the path is traversed during a back edge traversal. Unlike the S&H algorithm whose linked list contain only P-nodes (by flipping C-nodes to the correct side and eliminating the inner part), a path between any two consecutive delimiters in our linked list can be a general PC-tree containing

C-nodes whose *correct* sides are yet to be determined. The advantage of this design is that it allows essential nodes of these C-nodes to remain unidentified until more back edges are included later on. A C-node on a linked list is deleted whenever its correct side has been determined.

6 Complexity Analysis

The complexity of the algorithm can be analyzed as follows. The major operation is the back edge traversal. Minor operations involve delimiter list construction, boundary path composition and etc., which can be easily argued to be linear. Back edge traversal involves both tree edge traversal and the link traversal. We separate the edge traversal and the link traversal into two parts. First, consider the edge traversal assuming every node can find its parent in constant time (i.e. ignore the Find operation). The number of new nodes created (C-nodes) can be at most $O(n)$. Now, each tree edge can be traversed at most twice before it is either embedded inside some biconnected component, eliminated or become a link in the boundary cycle of some component. In the latter case, its traversal is counted as link traversal. Hence, the total number of tree edge traversal is $\mathrm{O}(m)$.

Next, consider the link traversal. The sole purpose of the parallel search is to identify the biconnected component that the current node belongs to by locating the closest delimiter. Once the delimiter is found, we can then eliminate all but the current segment. Since this new node u will become a delimiter for another C-node and either the left or the right side of it will eventually be eliminated in future iterations; or else they will never be traversed again. In other words, each link will be traversed at most a constant number of time. Hence, the total cost in traversing the linked lists is $\mathrm{O}(m)$. Therefore, the entire algorithm takes $\mathrm{O}(m)$ time.

Acknowledgement

We would like to thank the National Science Council for their generous support under Grant NSC 91-2213-E-001-011. I am also indebted to my research assistant, Kevin Mai, for his endeavor in the LEDA programming for the project.

References

1. K. S. Booth and G. S. Lueker, Testing the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comput. Syst. Sci. **13** (1976) 335–379
2. J. Boyer and W. Myvold, Stop minding your P's and Q's: A simplified $O(n)$ embedding algorithm. Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (1999) 140–149
3. J. Cai, X. Han and R. E. Tarjan, An $O(m\log n)$-time algorithm for the maximal planar subgraph problem. SIAM J. Comput. **22** (1993) 1142-1162

4. N. Chiba, T. Nishizeki and S. Abe and T. Ozawa, A linear algorithm for embedding planar graphs using PQ-trees. J. Comput. Syst. Sci. **30** (1985) 54-76
5. J. E. Hopcroft and R. E. Tarjan, Efficient planarity testing. J. Assoc. Comput. Mach. **21** (1974) 549-568
6. D. Hristo, A Linear Algorithm for the Maximal Planar Subgraph Problem. Workshop on Algorithms and Data Structures (1995) 369-380
7. W. L. Hsu, Finding maximal planar subgraphs in linear time. Lecture Notes in Computer Science **1004** (1995)
8. W. L. Hsu, PC-trees vs. PQ-trees. Lecture Notes in Computer Science **2108** (2001) 207-217
9. W. L. Hsu and R. McConnell, PQ Trees, PC Trees and Planar Graphs. In: Dinesh P. Mehta and Sartaj Sahni (eds) Handbook of Data Structures and Applications (2004)
10. R. Jayakumar, K. Thulasiraman and M. Swamy, On $O(n^2)$ algorithms for graph planarization. IEEE Transactions on Computer-Aided Design **8** (1989) 257-267
11. M. Junger, S. Leipert and P. Mutzel, A note on computing a maximal planar subgraph using PQ-Trees. IEEE Transactions on Computer-Aided Design **17** (1998) 609-612
12. G. Kant, An $O(n^2)$ maximal planarization algorithm based on PQ-trees. Technical Report RUU-CS-92-03, Department of Computer Science, Utrecht Unvesity. (1992)
13. A. Lempel, S. Even and I. Cederbaum, An algorithm for planarity testing of graphs. In P. Rosenstiehl, Gordon and Breach (eds.) Theory of Graphs, New York (1967) 215-232
14. K. Mehlhorn, Graph algorithms and NP-completeness. Data Structure and Algorithms **2** (1984) 93-122
15. K. Mehlhorn and P. Mutzel, On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. Algorithmica **16** (1996) 233–242
16. J. Small, A unified approach of testing, embedding and drawing planar graphs. Proc. ALCOM International Workshop on Graph Drawing, Sevre, France (1993)
17. W. K. Shih and W. L. Hsu, A simple test for planar graphs. Proceedings of the International Workshop on Discrete Math. and Algorithms, University of Hong Kong (1993) 110-122
18. W. K. Shih and W. L. Hsu, A new planarity test. Theoretical Computer Science **223** (1999) 179-191
19. H. Stamm-Wilbrandt, A simple linear-time algorithm for embedding maximal planar graphs. Proc, ALCOM International Workshop on Graph Drawing, Sere, France. (1993)
20. R. Thomas, Planarity in linear time – Lecture Notes. Georgia Institute of Technology (1997)
21. S. G. Williamson, Depth-first search and Kuratowski subgraphs. J. ACM **31** (1984) 681-693

Algorithms for Finding Distance-Edge-Colorings of Graphs

Takehiro Ito, Akira Kato, Xiao Zhou, and Takao Nishizeki

Graduate School of Information Sciences, Tohoku University,
Aoba-yama 6-6-05, Sendai, 980-8579, Japan
{take,akira}@nishizeki.ecei.tohoku.ac.jp, {zhou,nishi}@ecei.tohoku.ac.jp

Abstract. For a bounded integer ℓ, we wish to color all edges of a graph G so that any two edges within distance ℓ have different colors. Such a coloring is called a distance-edge-coloring or an ℓ-edge-coloring of G. The distance-edge-coloring problem is to compute the minimum number of colors required for a distance-edge-coloring of a given graph G. A partial k-tree is a graph with tree-width bounded by a fixed constant k. We first present a polynomial-time exact algorithm to solve the problem for partial k-trees, and then give a polynomial-time 2-approximation algorithm for planar graphs.

1 Introduction

We denote by $G = (V, E)$ a graph with vertex set V and edge set E. An ordinary *edge-coloring of a graph* G is to color all edges of G so that any adjacent edges have different colors. For two vertices u and v, we denote by $\mathrm{dist}(u, v)$ the *distance between* u *and* v in G, that is, the number of edges in a shortest path between u and v in G. For two edges $e = (u, v)$ and $e' = (u', v')$, the *distance between* e *and* e' in G is defined as follows:

$$\mathrm{dist}(e, e') = \min\{\mathrm{dist}(u, u'), \mathrm{dist}(u, v'), \mathrm{dist}(v, u'), \mathrm{dist}(v, v')\}.$$

For a given bounded nonnegative integer ℓ, we wish to color all edges of G so that any two edges e and e' with $\mathrm{dist}(e, e') \leq \ell$ have different colors. Such a coloring is called a *distance-edge-coloring* or an *ℓ-edge-coloring* of G. Thus a 0-edge-coloring is merely an ordinary edge-coloring, and a 1-edge-coloring is a "strong edge-coloring" [13, 14]. The *ℓ-chromatic index* $\chi'_\ell(G)$ of G is the minimum number of colors required for an ℓ-edge-coloring of G. The *distance-edge-coloring problem* or the *ℓ-edge-coloring problem* is to compute the ℓ-chromatic index $\chi'_\ell(G)$ of a given graph G. For example, the graph G in Fig. 1 has a 1-edge-coloring with six colors $c_1, c_2, \cdots, c_6$, and one can easily observe that $\chi'_1(G) = 6$. The coloring is of course a 0-edge-coloring, but is not a 2-edge-coloring.

Since the edge-coloring problem is NP-hard [12], the ℓ-edge-coloring problem is NP-hard in general and hence it is very unlikely that the ℓ-edge-coloring problem can be efficiently solved for general graphs. A partial k-tree is a graph with tree-width bounded by a fixed constant k. The class of partial k-trees is

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 798–807, 2005.

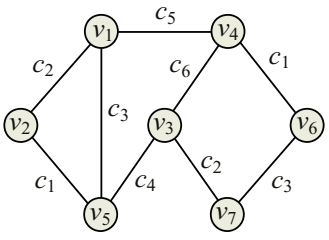


Fig. 1. A 1-edge-coloring of a partial 3-tree G with six colors.

fairly large, and includes trees, outerplanar graphs, series-parallel graphs, etc. It is known that many combinatorial problems can be solved very efficiently for partial k-trees even if the problems are NP-hard for general graphs [2, 3, 8–10]. Such classes of problems have been characterized in terms of "forbidden subgraphs" or "extended monadic second-order logic" [2, 3, 8–10]. The ℓ-edge-coloring problem does not belong to such a class of the "maximum or minimum subgraph problems" [10]. The ℓ-edge-coloring problem is indeed one of the "edge-covering problems" which, as mentioned in [8], does not appear to be efficiently solved for partial k-trees. However, the following two results have been known. First, the ordinary edge-coloring problem can be solved in linear time for partial k-trees [16]. Second, the 1-edge-coloring problem can be solved in polynomial time for partial k-trees [14].

A vertex version of the distance-edge-coloring problem has been studied for partial k-trees and planar graphs. For a given bounded nonnegative integer ℓ, the *distance-vertex-coloring* or *ℓ-vertex-coloring* is to color all vertices of a graph G so that any two vertices u and v with $\mathrm{dist}(u, v) \leq \ell$ have different colors. The distance-vertex-coloring problem, which finds an ℓ-vertex-coloring of a given graph with the minimum number of colors, can be solved in polynomial time for partial k-trees [15]. There is a polynomial-time 2-approximation algorithm for the distance-vertex-coloring problem on planar graphs [1]. The distance-edge-coloring problem for a graph G can be reduced to an ordinary vertex-coloring problem for a new graph G' obtained from G by some operations. However, G' is not always a partial k-tree or a planar graph even if G is a partial k-tree or a planar graph.

In this paper we first give a polynomial-time exact algorithm to solve the ℓ-edge-coloring problem for partial k-trees. More precisely, we give an algorithm to examine whether a partial k-tree G has an ℓ-edge-coloring with a given number α of colors in time $O\big(n(\alpha+1)^{2^{2(k+1)(\ell+1)+1}}\big)$, where n is the number of vertices in G. Remember that $k, \ell = O(1)$. One may assume without loss of generality that α is smaller than the number m of edges in G; otherwise, G has a trivial ℓ-edge-coloring with α colors. Thus the ℓ-edge-coloring problem can be solved in polynomial time. Our algorithm takes linear time if α is a fixed constant. We then give a polynomial-time 2-approximation algorithm for the ℓ-edge-coloring problem on planar graphs.

2 Terminology and Definitions

In this section we give some definitions. An edge joining vertices u and v is denoted by (u, v). We denote by n and m the number of vertices and edges in G, respectively, and assume that k is a bounded positive integer.

A *k-tree* is defined recursively as follows [5]:

(1) A complete graph with $k+1$ vertices is a k-tree.
(2) If G is a k-tree and k vertices induce a complete subgraph of G, then a graph obtained from G by adding a new vertex and joining it with each of the k vertices is a k-tree.

Every subgraph of a k-tree is called a *partial k-tree*. Thus a partial k-tree $G = (V, E)$ is a simple graph, and $m < kn$.

Figure 2 illustrates a process of generating 3-trees. The graph in Fig. 1 is indeed a partial 3-tree since it is a subgraph of the last 3-tree in Fig. 2.

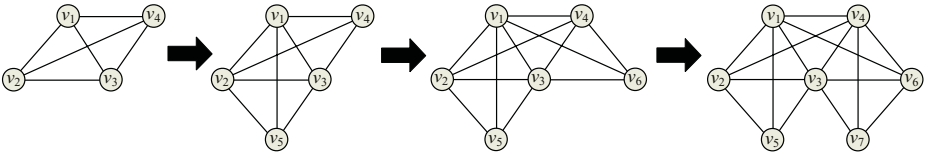


Fig. 2. A process of generating 3-trees.

A binary tree $T = (V_T, E_T)$ is called a *tree-decomposition of a partial k-tree* $G = (V, E)$ if T satisfies the following conditions (a)–(e):

(a) every node $X \in V_T$ of T is a subset of V, and $|X| \leq k+1$;
(b) $\bigcup_{X \in V_T} X = V$;
(c) for each edge $e = (u, v)$ of G, T has a leaf $X \in V_T$ such that $u, v \in X$;
(d) if node X_q lies on the path in T from node X_p to node X_r, then $X_p \cap X_r \subseteq X_q$; and
(e) each internal node X_i of T has exactly two children, say X_L and X_R, and either $X_i = X_L$ or $X_i = X_R$.

We will use notions leaf, node, child, and root in their usual meaning. Figure 3 illustrates a tree-decomposition T of the partial 3-tree in Fig. 1. Note that $V_T = \{X_0, X_1, \cdots, X_6\}$. We always denote by X_0 the root of a tree-decomposition T.

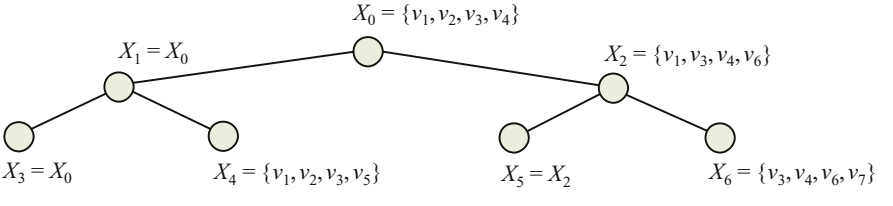


Fig. 3. Tree-decomposition of the partial 3-tree in Fig. 1.

Since a tree-decomposition T of a partial k-tree G can be found in linear time [6], we may assume that a partial k-tree G and its tree-decomposition T are given. The number of nodes of T constructed by the algorithm in [6] is $O(n)$.

By the condition (c) of a tree-decomposition, for every edge $e = (u, v) \in E$, there is at least one leaf X of T such that $u, v \in X$. We choose one of such leaves as the *representative* of the edge e, and denote it by $\mathrm{rep}(e)$. Each node X_i of T corresponds to a subgraph $G_i = (V_i, E_i)$ of G. The vertex set V_i and edge set E_i of G_i are recursively defined as follows:

(i) if X_i is a leaf of T, then $V_i = X_i$ and $E_i = \{e \in E \mid \mathrm{rep}(e) = X_i\}$; and

(ii) if X_i is an internal node of T, the left child X_L of X_i corresponds to a subgraph $G_L = (V_L, E_L)$ of G, and the right child X_R corresponds to $G_R = (V_R, E_R)$, then $V_i = V_L \cup V_R$ and $E_i = E_L \cup E_R$, and hence G_i is a union of two graphs G_L and G_R.

Note that $E_L \cap E_R = \emptyset$. Clearly $G = G_0$ for the root X_0 of T. The condition (d) of a tree-decomposition implies that $V_L \cap V_R = X_L \cap X_R \subseteq X_i$ [11].

3 Algorithm for Partial k-Trees

The main result of this section is the following theorem.

Theorem 1. *Let G be a partial k-tree, let ℓ be a bounded nonnegative integer, and let α be a positive integer. Then it can be examined in time $O\big(n(\alpha+1)^{2^{2(k+1)(\ell+1)+1}}\big)$ whether G has an ℓ-edge-coloring with α colors.*

The number α is not assumed to be a fixed constant, but can be assumed to be smaller than the number m of edges in G. Therefore, using a binary search technique, one can compute the ℓ-chromatic index $\chi'_\ell(G)$ of G by applying Theorem 1 for at most $\log_2 m$ values of $\alpha, 1 \le \alpha < m$. We thus have the following corollary.

Corollary 1. *The ℓ-chromatic index $\chi'_\ell(G)$ of a partial k-tree G can be computed in polynomial time.*

In the remainder of this section we give a proof of Theorem 1. From now on we call an ℓ-edge-coloring simply a *coloring*. Although we give an algorithm to examine whether a partial k-tree G has a coloring with α colors, it can be easily modified so that it actually finds a coloring of G with α colors if G has. Our idea is to extend techniques developed for the ordinary edge-coloring problem [5, 16] and the distance-vertex-coloring problem [15] to the ℓ-edge-coloring problem and is to reduce the size of a Dynamic Programming (DP) table to $O\big((\alpha+1)^{2^{2(k+1)(\ell+1)}}\big)$ by considering "counts" and "pair-counts."

Let $G = (V, E)$ be a partial k-tree, and let $T = (V_T, E_T)$ be a tree-decomposition of G. Let C be a set of α colors. For a node X_i of T, a mapping $f : E_i \to C$ is called an *entire coloring of* $G_i = (V_i, E_i)$ if $f(e) \neq f(e')$ for every pair of edges $e, e' \in E_i$ with $\mathrm{dist}(e, e') \le \ell$. Remember that $\mathrm{dist}(e, e')$ is the distance between e and e' in the entire graph G, not in the subgraph G_i. Thus

an entire coloring of G_i is a coloring of G_i, while a coloring of G_i is not always an entire coloring of G_i. However, a coloring of $G_0 = G$ is an entire coloring of G. Figures 4(a), (b) and (c) illustrate entire colorings of $G_0 = G$, G_1 and G_2, respectively, for the case $\ell = 1$.

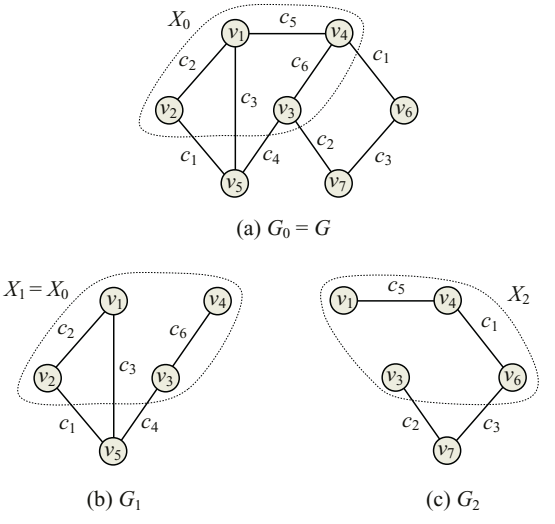


Fig. 4. (a) Colorings f_0 of $G = G_0$, (b) f_1 of G_1, and (c) f_2 of G_2.

For a vertex u and an edge $e = (v, w)$, the *distance between u and e* in G is defined as follows: $\mathrm{dist}(u, e) = \min\{\mathrm{dist}(u, v), \mathrm{dist}(u, w)\}$. Thus $\mathrm{dist}(u, e) = 0$ if u is an end-vertex of e.

For an entire coloring f of G_i, an integer j, $0 \le j \le \ell$, and a vertex $v \in X_i$, we define a set $D(f, j, v) \subseteq C$ as follows:

$$D(f, j, v) = \{c \in C \mid G_i \text{ has an edge } e \text{ such that } f(e) = c \text{ and } \mathrm{dist}(v, e) = j\}.$$

Thus $D(f, j, v)$ consists of all colors c that are assigned to edges e of G_i with $\mathrm{dist}(v, e) = j$. For example, $D(f_1, 0, v_1) = \{c_2, c_3\}$ and $D(f_1, 1, v_1) = \{c_1, c_4, c_6\}$ for the entire coloring f_1 of the graph G_1 in Fig. 4(b). Note that $\mathrm{dist}(v_1, v_4) = 1$ for the entire graph G depicted in Fig. 4(a).

For a node $X_i \in V_T$ of T, an entire coloring f of G_i, an integer j, $0 \le j \le \ell$, and a color $c \in C$, we define a set $Y(X_i; f, j, c) \subseteq X_i$ as follows:

$$Y(X_i; f, j, c) = \{v \in X_i \mid c \in D(f, j, v)\}.$$

Thus $Y(X_i; f, j, c)$ consists of all vertices v in X_i for which G_i has an edge e such that $f(e) = c$ and $\mathrm{dist}(v, e) = j$. For example, $Y(X_1; f_1, 0, c_6) = \{v_3, v_4\}$ and $Y(X_1; f_1, 1, c_6) = \{v_1\}$ for the entire coloring f_1 in Fig. 4(b).

We denote by 2^{X_i} the power set of X_i, and by $(2^{X_i})^{\ell+1}$ the direct product of $\ell + 1$ copies of 2^{X_i}. Thus, if $\mathbb{A} \in (2^{X_i})^{\ell+1}$, then $\mathbb{A}$ is an $(\ell + 1)$-tuple

$(A^0, A^1, \cdots, A^\ell)$ of sets $A^0, A^1, \cdots, A^\ell \subseteq X_i$. For an entire coloring f of G_i, we define a mapping $C_f : (2^{X_i})^{\ell+1} \to 2^C$ as follows:

$$C_f(\mathbb{A}) = \{c \in C \mid A^j = Y(X_i; f, j, c) \text{ for each } j,\ 0 \le j \le \ell\},$$

where $\mathbb{A} = (A^0, A^1, \cdots, A^\ell) \in (2^{X_i})^{\ell+1}$. For example, $C_{f_1}(\{v_3, v_4\}, \{v_1\}) = \{c_6\}$ and $C_{f_1}(\{v_3, v_4\}, \{v_1, v_2\}) = \emptyset$ for the entire coloring f_1 in Fig. 4(b). Probably $C_f(\mathbb{A}) = \emptyset$ for many $\mathbb{A} \in (2^{X_i})^{\ell+1}$. We call the mapping C_f the *color function of* f *on* X_i. We write $\mathcal{F}_f = \{C_f(\mathbb{A}) \mid \mathbb{A} \in (2^{X_i})^{\ell+1}\}$, then $\mathcal{F}_f$ is clearly a partition of the set C.

For a node X_i of T, we say that an entire coloring of G_i is *extensible* if it can be extended to a coloring of $G = G_0$ without changing the entire coloring of G_i. Both the entire coloring f_1 of G_1 in Fig. 4(b) and the entire coloring f_2 of G_2 in Fig. 4(c) are extensible because both can be extended to the coloring f_0 of G_0 in Fig. 4(a).

A mapping $\gamma : (2^{X_i})^{\ell+1} \to \{0, 1, \cdots, \alpha\}$ is called a *count on node* X_i. A count γ on X_i is defined to be *active* if G_i has an entire coloring f whose color function C_f satisfies $|C_f(\mathbb{A})| = \gamma(\mathbb{A})$ for each $\mathbb{A} \in (2^{X_i})^{\ell+1}$. Such a count γ is called the *count of the entire coloring* f. Since $|C| = \alpha$ and $\mathcal{F}_f$ is a partition of C, an active count γ satisfies $\sum\{\gamma(\mathbb{A}) \mid \mathbb{A} \in (2^{X_i})^{\ell+1}\} = \alpha$.

One can easily observe that the following lemma holds.

Lemma 1. *Assume that f and g are entire colorings of G_i for a node X_i of T, and that f and g have the same count. Then f is extensible if and only if g is extensible.*

Define an equivalence relation $\cong$ on the set of all entire colorings of G_i, as follows: $f \cong g$ if the entire colorings f and g of G_i have the same (active) count. Then each active count on X_i characterizes an equivalence class of entire colorings of G_i. Lemma 1 implies that either all the entire colorings in an equivalence class are extensible or none of them is extensible. Since $|X_i| \le k+1$, there are at most $(\alpha+1)^{2^{(k+1)(\ell+1)}}$ distinct counts $\gamma : (2^{X_i})^{\ell+1} \to \{0, 1, \cdots, \alpha\}$ on X_i. The main step of our algorithm is to compute a table of all active counts on each node of T from the leaves to the root X_0 of T by means of dynamic programming. From the table on the root X_0 one can easily know whether G has a coloring with α colors, as follows.

Lemma 2. *A partial k-tree G has a coloring with α colors if and only if the table on the root X_0 has at least one active count.*

We now describe an algorithm to examine whether a partial k-tree G has a coloring with the α colors $c_1, c_2, \cdots, c_\alpha$ in C.

We first compute the table of all active counts on each leaf X_i of T as follows:

(1) enumerate all mappings $f : E_i \to \{c_1, c_2, \cdots, c_j\}$, where $j = \min\{\alpha, |E_i|\}$;
(2) remove mappings that are not entire colorings of G_i; and
(3) compute all the active counts corresponding to entire colorings of G_i.

As a preprocessing, we compute $\mathrm{dist}(u, v)$ for all pairs of vertices u and v in the same leaf of T. This can be done in linear time; the proof is omitted in

this extended abstract, due to the page limitation. Since $|E_i| \leq k(k+1)/2$, the number of distinct mappings f enumerated in Step (1) above is at most $(\frac{k(k+1)}{2})^{k(k+1)/2} = O(1)$. Since the distances $\mathrm{dist}(u, v)$ have been computed, Step (2) above can be done in time $O(1)$ for each mapping f. Clearly Step (3) above can be done in time $O(1)$ for each entire coloring f. Thus one can compute the table on a leaf X_i of T in time $O(1)$. Since T has $O(n)$ leaves, the tables for all leaves can be computed in time $O(n)$.

We next compute all active counts on each internal node X_i of T from all active counts on its children X_L and X_R. Either $X_i = X_L$ or $X_i = X_R$ by the condition (e) of a tree-decomposition. Therefore, one may assume without loss of generality that $X_i = X_L$. A mapping $\rho : (2^{X_L})^{\ell+1} \times (2^{X_R})^{\ell+1} \to \{0, 1, \cdots, \alpha\}$ is called a *pair-count on* X_i. There are at most $(\alpha+1)^{2^{2(k+1)(\ell+1)}}$ distinct pair-counts. For an entire coloring f of G_i, we denote by $f_L = f|G_L$ the *restriction* of f to G_L: $f_L(e) = f(e)$ for each edge e of G_L. Similarly, we denote by $f_R = f|G_R$ the restriction of f to G_R. We denote by C_{f_L} the color function of f_L on X_L, and by C_{f_R} the color function of f_R on X_R. Then we define a pair-count ρ to be *active* if G_i has an entire coloring f such that $\rho(\mathbb{A}_L, \mathbb{A}_R) = |C_{f_L}(\mathbb{A}_L) \cap C_{f_R}(\mathbb{A}_R)|$ for each pair of $\mathbb{A}_L \in (2^{X_L})^{\ell+1}$ and $\mathbb{A}_R \in (2^{X_R})^{\ell+1}$. Such a pair-count ρ is called the *pair-count of the entire coloring* f *of* G_i. Thus, $\rho(\mathbb{A}_L, \mathbb{A}_R)$ is the number of colors $c \in C$ such that $A_L^j = Y(X_L; f_L, j, c)$ and $A_R^j = Y(X_R; f_R, j, c)$ for each j, $0 \leq j \leq \ell$, where $\mathbb{A}_L = (A_L^0, A_L^1, \cdots, A_L^\ell)$ and $\mathbb{A}_R = (A_R^0, A_R^1, \cdots, A_R^\ell)$. We now have the following lemma, whose proof is omitted due to the page limitation.

Lemma 3. *Let X_i be an internal node of T, and let X_L and X_R be the children of X_i. Then a pair-count ρ on X_i is active if and only if ρ satisfies the following Conditions (a) and (b):*

(a) if $\rho(\mathbb{A}_L, \mathbb{A}_R) \geq 1$, then $A_L^{j_1} \cap A_R^{j_2} = \emptyset$ for every pair of nonnegative integers j_1 and j_2 with $j_1 + j_2 \leq \ell$; and

(b) there is an active count γ_L on X_L such that

$$\gamma_L(\mathbb{A}_L) = \sum \{\rho(\mathbb{A}_L, \mathbb{A}) \mid \mathbb{A} \in (2^{X_R})^{\ell+1}\} \tag{1}$$

for each $\mathbb{A}_L \in (2^{X_L})^{\ell+1}$, and there is an active count γ_R on X_R such that

$$\gamma_R(\mathbb{A}_R) = \sum \{\rho(\mathbb{A}, \mathbb{A}_R) \mid \mathbb{A} \in (2^{X_L})^{\ell+1}\} \tag{2}$$

for each $\mathbb{A}_R \in (2^{X_R})^{\ell+1}$.

Using Lemma 3, we compute all active pair-counts ρ on X_i from all pairs of active counts γ_L on X_L and γ_R on X_R, as follows. There are at most $(\alpha+1)^{2^{2(k+1)(\ell+1)}}$ distinct pair-counts ρ on X_i. For each ρ of them, we examine whether ρ satisfies Conditions (a) and (b) in Lemma 3. For each pair-count ρ, one can know in time $O(1)$ whether ρ satisfies Condition (a), because there are at most $\ell^2 2^{2(k+1)(\ell+1)} = O(1)$ distinct pairs $(A_L^{j_1}, A_R^{j_2})$. On the other hand, for each pair-count ρ, one can know in time $O\big((\alpha+1)^{2^{(k+1)(\ell+1)+1}}\big)$ whether ρ satisfies Condition (b), because there are at most $\big((\alpha+1)^{2^{(k+1)(\ell+1)}}\big)^2 = (\alpha+1)^{2^{(k+1)(\ell+1)+1}}$ pairs of

active counts γ_L and γ_R, and one can know in time $O(1)$ for each of them whether it satisfies Eqs. (1) and (2). Thus all active pair-counts ρ on X_i can be found in time $O\big((\alpha+1)^{2^{2(k+1)(\ell+1)+1}}\big)$, since there are at most $(\alpha+1)^{2^{2(k+1)(\ell+1)}}$ distinct pair-counts ρ on X_i and $(\alpha+1)^{2^{(k+1)(\ell+1)+1}}(\alpha+1)^{2^{2(k+1)(\ell+1)}} \le (\alpha+1)^{2^{2(k+1)(\ell+1)+1}}$.

We then compute all active counts on an internal node X_i from all active pair-counts on X_i, as in the following Lemma 4, the proof of which is omitted due to the page limitation.

Lemma 4. *Assume that X_i is an internal node of T, X_L and X_R are the two children of X_i, and $X_i = X_L$. Then a count γ on X_i is active if and only if there exists an active pair-count ρ on X_i such that, for each $\mathbb{A} \in (2^{X_i})^{\ell+1}$,*

$$\gamma(\mathbb{A}) = \sum \rho(\mathbb{A}_L, \mathbb{A}_R), \tag{3}$$

where the summation above is taken over all $\mathbb{A}_L = (A_L^0, A_L^1, \cdots, A_L^\ell) \in (2^{X_L})^{\ell+1}$ and $\mathbb{A}_R = (A_R^0, A_R^1, \cdots, A_R^\ell) \in (2^{X_R})^{\ell+1}$ satisfying $A^j = (A_L^j \cup A_R^j) \cap X_i$ for each integer j, $0 \le j \le \ell$.

Using Lemma 4, we compute all active counts γ on X_i from all active pair-counts ρ on X_i. There are at most $(\alpha+1)^{2^{2(k+1)(\ell+1)}}$ distinct active pair-counts ρ. From each ρ of them we compute an active count γ by Eq. (3). This can be done in time $O(1)$ since $|A^j|, |A_L^j|, |A_R^j| \le k+1 = O(1)$ for each integer j, $0 \le j \le \ell$. We have thus shown that all active counts γ on X_i can be computed in time $O\big((\alpha+1)^{2^{2(k+1)(\ell+1)}}\big)$ from all active pair-counts ρ on X_i.

One can thus compute the DP table for an internal node X_i from the tables of the children X_L and X_R in time

$$O\big((\alpha+1)^{2^{2(k+1)(\ell+1)+1}} + (\alpha+1)^{2^{2(k+1)(\ell+1)}}\big) = O\big((\alpha+1)^{2^{2(k+1)(\ell+1)+1}}\big).$$

Since T has $O(n)$ internal nodes, one can compute the DP tables for all internal nodes in time $O\big(n(\alpha+1)^{2^{2(k+1)(\ell+1)+1}}\big)$.

From the DP table for the root X_0 one can know in time $O(1)$ by Lemma 2 whether G has a coloring with α colors.

This completes a proof of Theorem 1.

4 2-Approximation Algorithm for Planar Graphs

The main result of this section is the following theorem.

Theorem 2. *There is a polynomial-time 2-approximation algorithm for the distance-edge-coloring problem on planar graphs.*

In the remainder of this section, as a proof of Theorem 2, we give a polynomial-time algorithm to find an ℓ-edge-coloring of a given planar graph G with at most $2\chi'_\ell(G)$ colors. The approximation algorithm can be obtained by combining our algorithm in Section 3 with a general method for obtaining approximation algorithms for NP-complete problems on planar graphs [4].

The method [4] partitions the vertex set V of a planar graph $G = (V, E)$ into a number p of subsets $V_0, V_1, \cdots, V_{p-1}$ for some integer p so that every edge is between adjacent subsets or within the same subset, that is, if $(u, v) \in E$ and $u \in V_i$ then $v \in V_{i-1} \cup V_i \cup V_{i+1}$. Clearly, $\mathrm{dist}(u, v) \geq |i - j|$ if $u \in V_i$ and $v \in V_j$. Let

$$V' = \bigcup \{V_i \mid i \bmod 2(\ell+1) \leq \ell+1\}$$

and

$$V'' = (V - V') \cup \left(\bigcup \{V_i \mid i \bmod 2(\ell+1) = 0 \text{ or } \ell+1\} \right),$$

then both of the ends u and v of each edge $(u, v) \in E$ are contained in either V' or V''. Let $G' = (V', E')$ be the subgraph of G induced by V', and let $G'' = (V'', E'')$ be the subgraph of G such that $E'' = E - E'$. Then G' is a vertex-disjoint union of subgraphs H'_j, $0 \leq j \leq \lfloor p/2(\ell+1) \rfloor$; H'_j corresponds to $V_{2(\ell+1)j} \cup V_{2(\ell+1)j+1} \cup \cdots \cup V_{2(\ell+1)j+(\ell+1)}$. Every subgraph H'_j, $0 \leq j \leq \lfloor p/2(\ell+1) \rfloor$, is an $(\ell+2)$-outerplanar graph and hence is a partial $(3\ell+5)$-trees [7]. Since G' is a vertex-disjoint union of H'_j, $0 \leq j \leq \lfloor p/2(\ell+1) \rfloor$, G' is a partial $(3\ell+5)$-tree. Similarly, G'' is a vertex-disjoint union of subgraphs H''_j, $0 \leq j \leq \lfloor p/2(\ell+1) \rfloor$, and is a partial $(3\ell+5)$-tree; H''_j corresponds to $V_{2(\ell+1)j+\ell+1} \cup V_{2(\ell+1)j+\ell+2} \cup \cdots \cup V_{2(\ell+1)j+2(\ell+1)}$.

We now describe the approximation algorithm. We first compute $\mathrm{dist}(e, e')$ for all edges e and e' in the entire graph G in advance. This preprocessing can be done in time $O(n^2)$ by executing the breadth-first search with each vertex as a starting vertex. We then find an entire ℓ-edge-coloring of G' with the minimum number $\chi^*_\ell(G')$ of colors by using the polynomial-time algorithm in Section 3. In the entire ℓ-edge-coloring of G', any two edges e and e' with $\mathrm{dist}(e, e') \leq \ell$ must have different colors, where $\mathrm{dist}(e, e')$ is the distance between e and e' in the entire graph G, not in G'. Thus $\chi^*_\ell(G') \leq \chi'_\ell(G)$. Similarly, we find an entire ℓ-edge-coloring of G'' with the minimum number $\chi^*_\ell(G'')$ of colors, where $\chi^*_\ell(G'') \leq \chi'_\ell(G)$. One may assume that the colors for G' are different from the colors for G''. Combining the colorings of G' and G'', we finally obtain an ℓ-edge-coloring of G with $\chi^*_\ell(G') + \chi^*_\ell(G'') \leq 2\chi'_\ell(G)$ colors. This completes a proof of Theorem 2.

5 Conclusions

In this paper, we obtained two algorithms. The first algorithm is to examine whether a given partial k-tree G has an ℓ-edge-coloring with α colors in time $O\big(n(\alpha+1)^{2^{2(k+1)(\ell+1)+1}}\big)$, where n is the number of vertices in G and α is an arbitrary positive integer. Using the algorithm, one can compute the ℓ-chromatic index $\chi'_\ell(G)$ of G in polynomial time. Our algorithm takes linear time if α is a fixed constant. It is easy to modify the algorithm so that it actually finds an ℓ-edge-coloring of G with $\chi'_\ell(G)$ colors. The second algorithm is a polynomial-time 2-approximation algorithm for the distance-edge-coloring problem on planar graphs.

Many variants of the distance-edge-coloring problem can be solved for partial k-trees in polynomial time. Consider for example a problem in which, for a given set $L \subseteq \{0, 1, \cdots, \ell\}$, one wishes to color all edges of a graph G with the minimum number of colors so that every pair of edges e and e' with $\mathrm{dist}(e, e') \in L$ have different colors. Such a problem can be solved in polynomial time for partial k-trees similarly as the ℓ-edge-coloring problem.

Replace some of the edges in a partial k-tree by multiple edges. The resulting multigraph is called a *partial k-multitree*. One can easily extend our algorithms for partial k-trees and planar simple graphs to those for partial k-multitrees and planar multigraphs.

References

1. G. Agnarsson and M. M. Halldórson, Coloring powers of planar graphs, *SIAM J. Discrete Math.*, Vol. 16, pp. 651–662, 2003.
2. S. Arnborg, B. Courcelle, A. Proskurowski and D. Seese, An algebraic theory of graph reduction, *J. Assoc. Comput. Mach.*, Vol. 40, pp. 1134–1164, 1993.
3. S. Arnborg, J. Lagergren and D. Seese, Easy problems for tree-decomposable graphs, *J. Algorithms*, Vol. 12, pp. 308–340, 1991.
4. B. S. Baker, Approximation algorithms for NP-complete problems on planar graphs, *J. Assoc. Comput. Mach.*, Vol. 41, pp. 153–180, 1994.
5. H. L. Bodlaender, Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees, *J. Algorithms*, Vol. 11, pp. 631–643, 1990.
6. H. L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, *SIAM J. Computing*, Vol. 25, pp. 1305–1317, 1996.
7. H. L. Bodlaender, A partial k-arboretum of graphs with bounded treewidth, *Theoretical Computer Science*, Vol. 209, pp. 1–45, 1998.
8. R. B. Borie, R. G. Parker and C. A. Tovey, Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families, *Algorithmica*, Vol. 7 pp. 555–581, 1992.
9. B. Courcelle, The monadic second-order logic of graphs I: Recognizable sets of finite graphs, *Information and Computation*, Vol. 85, pp. 12–75, 1990.
10. B. Courcelle and M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs, *Theoretical Computer Science*, Vol. 109, pp. 49–82, 1993.
11. R. Diestel, Graph Theory, *Springer-Verlag, New York*, 1997.
12. I. Holyer, The NP-completeness of edge-coloring, *SIAM J. Computing*, Vol. 10, pp. 718–720, 1981.
13. M. Mahdian, On the computational complexity of strong edge coloring, *Discrete Applied Mathematics*, Vol. 118, pp. 239–248, 2002.
14. M. R. Salavatipour, A polynomial time algorithm for strong edge coloring of partial k-trees, *Discrete Applied Mathematics*, Vol. 143, pp. 285–291, 2004.
15. X. Zhou, Y. Kanari and T. Nishizeki, Generalized vertex-coloring of partial k-trees, *IEICE Trans. on Fundamentals of Electronics, Communication and Computer Sciences*, Vol. E83-A, pp. 671–678, 2000.
16. X. Zhou, S. Nakano and T. Nishizeki, Edge-coloring partial k-trees, *J. Algorithms*, Vol. 21, pp. 598–617, 1996.

On the Recognition of Probe Graphs of Some Self-Complementary Classes of Perfect Graphs

Maw-Shang Chang[1,⋆], Ton Kloks[3,⋆⋆], Dieter Kratsch[2], Jiping Liu[3,⋆⋆⋆], and Sheng-Lung Peng[4]

[1] Department of Computer Science and Information Engineering
National Chung Cheng University, Chiayi, Taiwan 621, R.O.C.
mschang@cs.ccu.edu.tw
[2] Université de Metz
LITA, 57045 Metz Cedex 01, France
kratsch@sciences.univ-metz.fr
[3] Department of Mathematics and Computer Science
The university of Lethbridge, Alberta, T1K 3M4, Canada
liu@cs.uleth.ca
[4] Department of Computer Science and Information Engineering
National Dong Hwa University, Hualien 974, Taiwan, R.O.C.
lung@csie.ndhu.edu.tw

Abstract. In this paper we consider the recognition of some probe graph classes. Given a class of graphs $\mathcal{G}$, a graph G is a probe graph of $\mathcal{G}$ if its vertices can be partitioned into a set $\mathbb{P}$ of *probes* and an independent set $\mathbb{N}$ of *nonprobes*, such that G can be extended to a graph of $\mathcal{G}$ by adding edges between certain nonprobes. We show that there are polynomial-time recognition algorithms for probe cographs, probe P_4-reducible graphs, probe P_4-sparse graphs, and probe splitgraphs.

1 Introduction

We will consider only finite simple graphs in which n and m are the number of vertices and edges of a graph, respectively. For a graph $G = (V, E)$ and a subset $S \subseteq V$ of vertices, we write $G[S]$ for the subgraph of G *induced* by S. For a vertex x we use $N(x)$ to denote the set $\{v \in V | (x, v) \in E\}$ and use $N[x]$ to denote $N(x) + x$. For a subset $W \subseteq V$ of vertices of a graph $G = (V, E)$ we write $G - W$ for the graph $G[V - W]$, *i.e.*, the subgraph induced by $V - W$. For a vertex x we write $G - x$ rather than $G - \{x\}$. For other conventions on graph-related notations we refer to any standard textbook. For graph classes not defined here we refer to [2, 8].

⋆ I thank the Institute of Information Science of Academia Sinica of Taiwan for its hospitality and support where part of this research took place

⋆⋆ Supported by the National Science Council under grant NSC 93–2811–M–002–004. I thank the LITA of the Université de Metz and the Department of Mathematics of the National Taiwan University

⋆⋆⋆ Corresponding author. Partially supported by the NSERC of Canada

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 808–817, 2005.

Probe interval graphs were introduced in [21, 23] to model certain problems in physical mapping of DNA when only partial data is available on the overlap of clones. Chapter 4 of [10] is dedicated to this class of graphs. In the biological application the partition of the vertex set into two sets, probes and nonprobes, is part of the input, and the problem is to add edges between some nonprobes to complete the graph into an interval graph. This problem was solved successfully in [18] and an $O(n^2)$ time algorithm was established. An alternative $O(n + m \log n)$ time algorithm appeared in [20]. The problem of recognizing probe interval graphs in polynomial-time when the partition of the vertex set is *not* part of the input was recently solved in [3]. Earlier attempts were made to gain a better understanding of the structure of these graphs by analyzing probe *chordal* graphs [9], where polynomial-time recognition algorithms were developed both in the case when a partition is a part of the input and for the situation when no partition is given. It is interesting to investigate other probe classes of graphs.

Definition 1. *Let $\mathcal{G}$ be a class of graphs. A graph G is a* probe graph of $\mathcal{G}$ *if its vertices can be partitioned into a set $\mathbb{P}$ and an independent set $\mathbb{N}$ such that G can be embedded into a graph $G' \in \mathcal{G}$ by adding edges between certain vertices of $\mathbb{N}$. The vertices of $\mathbb{P}$ are called* probes *and those of $\mathbb{N}$ are called* nonprobes.

If the partition of the vertices into probes and nonprobes is part of the input, then we call the graph a *partitioned probe graph of $\mathcal{G}$*. We call a graph $G' \in \mathcal{G}$ obtained from G by adding some edges between vertices of $\mathbb{N}$ an *embedding* of G. We have the following easy but useful observation.

Theorem 1 (Probe Sandwich Theorem). *Let $\mathcal{G}$ be a* self-complementary *class of graphs,* i.e., *$H \in \mathcal{G} \Leftrightarrow \overline{H} \in \mathcal{G}$. Let $G = (\mathbb{P} + \mathbb{N}, E)$ be a graph with a partition of its vertices into $\mathbb{P}$ and an independent set $\mathbb{N}$. Let G^* be obtained from $\overline{G}$ by removing all edges between vertices of $\mathbb{N}$. Then G is a partitioned probe graph of $\mathcal{G}$ if and only if G^* is in the same category.*

Another motivation for investigating probe self-complementary classes of graphs is the relation to perfect graphs. Lovász proved in 1972 that the class of perfect graphs[1] is self-complementary [19]. Considering probe graphs of classes of perfect graphs gives rise new classes, and it is interesting to see that many probe classes of perfect graphs are still perfect. If $G = (\mathbb{P} + \mathbb{N}, E)$ is a probe perfect graph, then the complement of every odd cycle of length at least five contains exactly two nonprobes, and these are connected by an edge in any embedding of G. This observation, and the Probe Sandwich Theorem 1 mentioned above, lead us to conjecture the following:

[1] A graph G is called *perfect* if for every induced subgraph the chromatic number equals its maximum clique size. For a nice appetizer on perfect graphs we refer to [11]. The *strong perfect graph theorem* (formulated by Berge), states that a graph is perfect if and only if it does not contain an induced *odd* cycle of length at least 5 or the complement of such a cycle. Recently this was proved to be correct [4]. Furthermore, an $O(n^9)$ time recognition algorithm for perfect graphs was announced [7]

Conjecture 1 (Probe Perfect Graph Conjecture). There exists a polynomial-time algorithm to test whether a partitioned graph $G = (\mathbb{P} + \mathbb{N}, E)$ is probe perfect.

Conjecture 2 (Strong Probe Perfect Graph Conjecture). There exists a polynomial-time algorithm to test whether a graph is probe perfect.

Remark 1. When a suitable embedding of a graph G into a perfect graph G' is given, a nice decomposition tree for G' can be obtained in polynomial time and this decomposition tree could be of use to solve NP-complete problems for G in polynomial-time. Here lies one of the motivations for studying probe perfect graph classes.

One of the merits of perfect graphs is that some of the 'basic' NP-complete problems such as CLIQUE, INDEPENDENT SET, CHROMATIC NUMBER, and CLIQUE COVER become solvable in polynomial-time when restricted to perfect graphs [1]. For probe classes of perfect graphs, we have the following theorem.

Theorem 2. *Let $\mathcal{G}$ be any class of perfect graphs. Let $\mathcal{PPG}$ be the class of partitioned probe graphs of $\mathcal{G}$. Then the* CLIQUE *problem can be solved in polynomial-time for all graphs in $\mathcal{PPG}$.*

Proof. Let $G = (\mathbb{P}+\mathbb{N}, E) \in \mathcal{PPG}$. By the recent algorithm that appeared in [7], there exists a polynomial-time algorithm to test whether a graph is perfect. Recall that, if a graph is perfect the CLIQUE problem is tractable in polynomial-time, via Lovász theta function, see, *e.g.*, pp. 325–356 in [1].
Observe that $\omega(G)$ can be computed as follows: For every vertex $x \in \mathbb{N}$, compute the maximum clique size of $G[N[x]]$, which is a perfect graph. Also compute the maximum clique size of $G[\mathbb{P}]$, which is likewise perfect. Then $\omega(G)$ will be the maximal value of these, and thus it can be computed in polynomial-time. □

Remark 2. Notice that the chromatic number of such a graph G is at most $\omega(G)+1$, since, after coloring $G[\mathbb{P}]$ with $\omega(G[\mathbb{P}])$ colors we can color the vertices of $\mathbb{N}$ using only one additional color.

In our paper we expand the research into probe graph classes by investigating probe cographs, probe P_4-reducible graphs, probe P_4-sparse graphs, and probe splitgraphs. The general strategy is to investigate the partitioned case first, then to deal with the unpartitioned case by exhibiting a polynomial number of feasible partitions.

2 Probe Cographs

A *cograph* is a graph without an induced P_4, *i.e.*, an induced path with 4 vertices. By now there are many characterizations known and in the literature various characterizations of the class are used to define the class. There is a wide variety of linear time cograph recognition algorithms. To mention just a few, see, *e.g.*, [6, 12]. For our purpose, the following portrayal of the class will make the grade [5].

Theorem 3. *[5] Cographs can be characterized as follows:*

1. *A graph consisting of a single vertex is a cograph.*
2. *Let G_1 and G_2 be cographs. Then the* join *of G_1 and G_2, obtained by making every vertex of G_1 adjacent to every vertex of G_2 is again a cograph.*
3. *Let G_1 and G_2 be cographs. Then the* union *of G_1 and G_2 is again a cograph.*

To strike up an acquaintance with the class of probe cographs we mention some bagatelles. The proofs of these observations are omitted because of limited space.
Observations. *Let G be a probe cograph. Then:*

a. *Every induced $2K_2$ in a probe cograph remains an induced $2K_2$ in every embedding.*
b. *If G contains an induced P_5, say $P = [u, v, w, x, y]$, then vertices u, w, and y must be nonprobes.*
c. *G has no induced P_{k+1} and C_k for any $k \geq 5$. Other forbidden induced subgraphs include parachute, domino, co-rising sun, the parapluie, and plenty of others. We refer to [2] for depicting these graphs.*
d. *A probe cograph is weakly chordal [13]; hence it is perfect.*
e. *If G is a probe cograph then its* clique-width *is at most 4.*

The following observations enable us to reduce the recognition problem in case the graph or its complement is disconnected.

Theorem 4. 1. *Suppose $G = (V, E)$ is disconnected. Then G is a probe cograph if and only if the subgraph induced by every connected component is a probe cograph.*
2. *Let $G = (V, E)$ be a graph whose complement $\overline{G}$ is disconnected. Let $C_1, C_2, \ldots, C_k$ be the components of $\overline{G}$. Then G is a probe cograph if and only if all induced subgraphs $G[C_1], G[C_2], \ldots, G[C_k]$ are cographs except possibly one which is a probe cograph.*

Applying Theorem 1 and Theorem 4, we have the following result.

Theorem 5. *There exists an $O(n^3)$ time algorithm to test whether a partitioned graph $G = (\mathbb{P} + \mathbb{N}, E)$ is a probe cograph.*

Proof. Let $G = (\mathbb{P} + \mathbb{N}, E)$ be a partitioned graph. Since there exists linear time recognition algorithm for cographs and using an adjacency matrix for the graph G, we can find a representation for $\overline{G}$ in $O(n^2)$ time; then Theorem 4 reduces the problem to the case when both G and $\overline{G}$ are connected in time $O(n^2)$.
Suppose that G is a probe cograph and let H be a valid embedding of G into a cograph. Since G is connected, and since H is obtained from G by adding some edges to it, also H will be connected. By Theorem 3, H will be the *join* of two cographs H_1 and H_2. Obviously, $\overline{H}$ is disconnected and the union of $\overline{H_1}$ and $\overline{H_2}$. The graph G^* is obtained by deleting some edges from H; hence it is disconnected. Again using the adjacency matrix we find a representation of G^* in $O(n^2)$ time. By the Probe Sandwich Theorem, G is a probe cograph if and

only if G^* is a probe cograph. Let $C_1, \ldots, C_k$ be the components of G^* and check whether each $G[C_i]$ is a probe cograph recursively. If G^* is connected, then G is not a probe cograph.
We have described the recognition algorithm recursively and shown its correctness. The algorithm first checks whether the input graph is a cograph, which can be done in $O(n+m)$ time. If it is not, then it takes $O(n^2)$ time to reduce the problem to test whether each component of G, $\overline{G}$, or G^* induces a probe cograph. There are at most $O(n)$ tests with reductions. Therefore the running time of the algorithm is $O(n^3)$. □

Remark 3. The proof shows that each probe cograph has a decomposition tree where the leaves are the vertices of the graph and the internal nodes are of *three* types: either a join node, a union node, or a "sandwich join" node. This decomposition tree can be built in $O(n^3)$ time.

In the rest of this section we show that there is a polynomial time algorithm that finds a suitable nonprobe set $\mathbb{N}$ for an unpartitioned probe cograph.

Theorem 6. *Let G and $\overline{G}$ be connected and assume that G is not a cograph. Then G is a probe cograph if and only if there are two non–adjacent vertices x and y in G such that G is a probe cograph with probe set $\mathbb{P} = N(x) + N(y)$ and nonprobe set $\mathbb{N} = V - \mathbb{P}$.*

Proof. Assume G is a probe cograph with an embedding G'. We may assume that G' is the *join* of two graphs G'_1 and G'_2 since G' is obtained from the connected graph G by adding some edges.
Let $\mathbb{N}_i$ and $\mathbb{P}_i$ be the probes and nonprobes in G'_i, $i = 1, 2$. We may assume $\mathbb{N}_1 \neq \varnothing$ and $\mathbb{N}_2 \neq \varnothing$ since $\overline{G}$ is connected. Take $x \in \mathbb{N}_1$ and $y \in \mathbb{N}_2$. □

Using Theorem 4 and Theorem 6 we have the following theorem.

Theorem 7. *There is an $O(n^5)$ time algorithm to test whether G is a probe cograph.*

Proof. Using Theorem 4 we can reduce the problem to the case where both G and $\overline{G}$ are connected. This reduction can obviously be done within the claimed time bound.
By Theorem 6 there exists a collection of $O(n^2)$ feasible partitions of the form $\mathbb{P} = N(x) + N(y)$ and $\mathbb{N} = V - \mathbb{P}$. For every such pair of vertices, it takes $O(n+m)$ to obtain the actual partition into probes $\mathbb{P}$ and nonprobes $\mathbb{N}$. Each of these $O(n^2)$ partitions can be tested using the algorithm of Theorem 5 in $O(n^3)$ time. Therefore, the overall time complexity for testing if G is a probe cograph is $O(n^2(n^3 + n + m)) = O(n^5)$. □

3 Probe P_4-Reducible Graphs

From this section, most of the proofs are omitted because of space limitations.

Definition 2. *A graph G is* P_4-reducible *if every vertex belongs to at most one induced P_4 of G.*

Definition 3. *An* ornament *in a graph G is an induced P_4, $P = [a, b, c, d]$ such that every vertex of $G - P$ is adjacent to b and to c and non-adjacent to a and to d.*

P_4-reducible graphs were defined in [15] and in this paper appeared also the following characterization which led to a linear time recognition algorithm.

Theorem 8 ([15]). *A graph G is P_4-reducible if and only if for every induced subgraph H of G, exactly one of the following conditions is satisfied:*

1. *either H or $\overline{H}$ is disconnected, or*
2. *there is a unique ornament.*

Remark 4. Notice that the class of P_4-reducible graphs is self complementary. Since P_4-reducible graphs have no induced house, it follows that probe P_4-reducible graphs are weakly chordal, and hence perfect.

We first consider the partitioned case: a graph $G = (\mathbb{P} + \mathbb{N}, E)$ with a partition of V into probes $\mathbb{P}$ and nonprobes $\mathbb{N}$ is given.

Lemma 1. *Let $G = (\mathbb{P} + \mathbb{N}, E)$ be a partitioned graph. Assume P is a set of 4 vertices in G that can be made into an ornament of G by adding some edges between vertices of $\mathbb{N}$. Then G is probe P_4-reducible if and only if $G - P$ is probe P_4-reducible.*

The next lemma deals with the case that G can be embedded into a P_4-reducible graph G' such that $\overline{G'}$ is disconnected.

Lemma 2. *Let $G = (\mathbb{P} + \mathbb{N}, E)$ be a connected partitioned graph. Let G^* be the graph obtained from $\overline{G}$ by deleting all edges between vertices of $\mathbb{N}$. Let $C_1, \ldots, C_k$ be the components of G^*, and let*

$$\mathbb{N}_i = (C_i \cap \mathbb{N}) \ \wedge \ (\mathbb{P}_i = C_i \cap \mathbb{P})$$

Then G is probe P_4-reducible if and only if each $G[\mathbb{P}_i + \mathbb{N}_i]$ is probe P_4-reducible with a vertex partition into probes $\mathbb{P}_i$ and nonprobes $\mathbb{N}_i$.

Theorem 9. *There exists a polynomial-time algorithm that checks if a graph $G = (\mathbb{P} + \mathbb{N}, E)$ is a partitioned probe P_4-reducible graph.*

Proof. If G is disconnected, then G is probe P_4-reducible if and only if every component is probe P_4-reducible.
If $G = (\mathbb{P} + \mathbb{N}, E)$ is a connected probe P_4-reducible graph, then either Lemma 1 or Lemma 2 with $k \geq 2$ applies. This can be tested in polynomial-time. If neither lemma applies then G is not a partitioned probe P_4-reducible graph. □

We now describe the recognition algorithm for the unpartitioned case.

i. Assume that G' is a *disconnected* embedding. Then G is also disconnected, and G is probe P_4-reducible if and only if every component is probe P_4-reducible. Henceforth assume that G is connected.
ii. Assume that G' is an embedding such that $\overline{G'}$ is disconnected. Identically to the proof of Theorem 6 it can be shown that in this case there must exist non-adjacent vertices x and y in G such that choosing $\mathbb{P} = N(x) + N(y)$ and $\mathbb{N} = V - \mathbb{P}$ provides a valid partition.
iii. Assume that G' is an embedding with an ornament $P = [a, b, c, d]$. Notice that, if G *itself* has some ornament P', then G is probe P_4-reducible if and only if $G - P'$ is P_4-reducible. Hence, we may assume that P is not an ornament of G. Since G is connected, we may assume that $a, d \in \mathbb{P}$ in some valid partition if this exists. We consider three possibilities:
   a. There is a valid partition with $b \in \mathbb{P}$ and $c \in \mathbb{N}$. Hence b is adjacent in G to all vertices of $V - P$ and $\mathbb{N} = (V - P - N(c)) + c$.
   b. There is a valid partition with $b, c \in \mathbb{N}$. Hence b and c are not adjacent in G. Then $N(b) - P = N(c) - P = \mathbb{P} - \{a, d\}$.
   c. If $b, c \in \mathbb{P}$, then P is an ornament in G which contradicts our assumption.

By the discussion above we obtain:

Theorem 10. *There exists a polynomial-time algorithm to test whether a graph G is a probe P_4-reducible graph and produce a valid embedding if this is the case.*

4 Probe P_4-Sparse Graphs

Hoàng introduced P_4-sparse graphs [14].

Definition 4. *A graph G is* P_4-sparse *if no set of 5 vertices induces more than one P_4.*

In [16] Jamison and Olariu characterized P_4-sparse graphs using spiders.

Definition 5. *A graph G is a* spider *if there is a partition of the vertices into three sets S, K, and R, satisfying:*

1. *S is an independent set, K is a clique, and $|S| = |K| \geq 2$,*
2. *every vertex of R is adjacent to every vertex of K and to no vertex of S,*
3. *there is a bijection f between S and K such that* either $\forall_{x \in S} N(x) = \{f(x)\}$, or $\forall_{x \in S} N(x) = K - f(x)$. *In the first case, G is called a* thin *spider and in the second case G is a* thick *spider.*

The set R is called the head *of the spider.*

Theorem 11 ([16]). *A graph is P_4-sparse if and only if for every induced subgraph H exactly one of the following conditions is satisfied:*

1. *either H or $\overline{H}$ is disconnected, or*
2. *H is isomorphic to a spider.*

Notice that Theorems 4, 6, and Lemma 2 can be "translated" into similar statements for P_4-sparse graphs. Due to space limitations we have to omit the precise description.
In the following lemma we show how to check whether a partitioned graph G can be embedded into a *thin spider* by adding some edges in $\mathbb{N}$.

Lemma 3 (Thin spider embedding). *Assume $G = (\mathbb{P} + \mathbb{N}, E)$ is connected and partitioned.*

1. *Assume there exists a vertex $x \in \mathbb{P}$ and a* pendant *vertex $y \in N(x)$ such that V partitions into a set $S = y + (V - N[x])$ of pendants, $K = N(S)$, and $R = V - K - S$, and assume that this partition can be completed into a thin spider by adding edges to $\mathbb{N}$. Then G is a probe P_4-sparse if and only if $G[R]$ is probe P_4-sparse with the induced partition into probes and nonprobes.*
2. *Let S be the set of pendant vertices that are probes. Let $K = N(S)$ and $R = V - K - S$. Assume G can be embedded into a thin spider with this partition. Then G is a probe P_4-sparse if and only if $G[R]$ is probe P_4-sparse with the induced partition.*
3. *Otherwise, G is not a probe thin spider.*

Lemma 4 (Thick spider embedding). *Let $G = (\mathbb{P} + \mathbb{N}, E)$ be a partitioned graph. Let G^* be the graph obtained from $\overline{G}$ by removing all edges between vertices of $\mathbb{N}$. Then G is a probe thick spider if and only if G^* is a probe thin spider.*

Proof. The class of P_4-sparse graphs is self-complementary. The complement of a thin spider is a thick spider. The claim now follows immediately from the Probe Sandwich Theorem 1. □

Theorem 12. *There exists a polynomial-time algorithm that checks if a partitioned graph $G = (\mathbb{P} + \mathbb{N}, E)$ is probe P_4-sparse.*

Now we have arrived at the unpartitioned recognition problem.

Theorem 13. *There exists a polynomial-time algorithm to test whether a graph G is a probe P_4-sparse graph and produce a valid embedding if this is the case.*

Proof. The case where G or G^* is disconnected is easy. Notice that if there is an embedding with a disconnected complement, then also G^* is disconnected. Lemmas 3 and 4 deal with the case where G is embedded into a spider. The proofs of these lemmas show that only a polynomial number of partitions need to be checked. □

5 Probe Splitgraphs

Recall that a graph $G = (V, E)$ is a *splitgraph* if its vertices can be partitioned into a clique C and an independent set S. We use $G = (C, S, E)$ to denote a splitgraph. We observe that probe splitgraphs are perfect.

Lemma 5. *Probe splitgraphs are perfect.*

Assume that $G' = (C, S, E')$ is an embedding of a probe splitgraph $G = (V, E)$. Let $\mathbb{N}$ be the set of nonprobes, i.e., $\mathbb{N}$ is an independent set in G, every edge in $E' - E$ connects two vertices in $\mathbb{N}$, and every vertex in $\mathbb{N}$ is adjacent to an edge in $E' - E$. Let $\mathbb{N}_C$ and $\mathbb{N}_S$ denote $\mathbb{N} \cap C$ and $\mathbb{N} \cap S$, respectively. Let $\mathbb{P}$ and $\mathbb{P}_C$ denote $V - \mathbb{N}$ and $C - \mathbb{N}_C$, respectively. There exists an embedding of G such that $\mathbb{N}_S$ is empty. Thus, to test whether an unpartitioned graph is a probe splitgraph, it is sufficient to hit upon $\mathbb{N}_C$ and add the necessary edges to turn it into a clique.

Theorem 14. *A graph G is a probe splitgraph if and only if G is a splitgraph or if there exists a clique K and one vertex $\kappa \in K$ such that $G - K$ is bipartite and there is a bipartition of $G - K$ such that every vertex of one color class is adjacent to every vertex of $K - \kappa$ and not adjacent to κ.*

The recognition algorithm for unpartitioned probe split graphs is now lucid. We assume that G is not a splitgraph, then $\mathbb{N}_C \neq \varnothing$. If G is bipartite (*i.e.*, if $\omega(G) = \chi(G) \leq 2$) then we complete it into a splitgraph by making a clique of one of the two color classes.
Next assume $\omega(G) \geq 3$ and G is probe split with nonprobes $\mathbb{N} = \mathbb{N}_C$ and probes $\mathbb{P}$. Let K be a maximal clique of G. Then K consists of a subset of $C - \mathbb{N}_C$ and either one vertex $\kappa \in \mathbb{N}_C$, or one vertex $s \in S$, or both. For each $w \in \mathbb{N}$ either $(C - \mathbb{N}_C) \cup \{w\}$ or $(C - \mathbb{N}_C) \cup \{w, s\}$ for some $s \in S$ is a maximal clique of G. Furthermore, it is not hard to see that G has $O(n^2)$ maximal cliques. Our algorithm generates all maximal cliques of G in time $O(n^5)$ using an algorithm given in [22]. Then it checks for every maximal clique K of G all possible deletions of one or two vertices to obtain a candidate for $\mathbb{P}_C$. The desired partition of the bipartite remaining part as requested in Theorem 14 can be checked in $O(n+m)$ time. There are $O(n^2)$ maximal cliques and n^2 possible deletions of vertices; thus whether G is a probe splitgraph can be checked in time $O(n^4(n+m))$.

Theorem 15. *There exists a polynomial-time algorithm to recognize probe splitgraphs.*

Acknowledgments

We thank Ross McConnell for preliminary discussions concerning some of the topics of this paper. We thank Gérard Chang for fruitful discussions concerning several parts of this paper. We thank Alexander Schrijver and Rolf Niedermeier for ongoing discussions concerning the probe perfect graph conjectures. Finally we are grateful to David Chandler for his careful proofreading.

References

1. Berge, C. and C. Chvatal ed., *Topics on Perfect Graphs*, Annals of Discrete Mathematics **21**, 1984.

2. Brändstadt, A., V. B. Le, and J. P. Spinrad, *Graph classes: A survey* SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 1999.
3. Chang, G. J., A. J. J. Kloks, J. Liu, and S. L. Peng, The PIGs full monty—a floor show of minimal separators, to appear in STACS'2005, LNCS 3404, pp. 521–532.
4. Chudnovsky, M., P. Seymour, N. Robertson, and R. Thomas, The strong perfect graph theorem. Manuscript 2002.
5. Corneil, D. G., H. Lerchs, and L. Stewart-Burlingham, Complement reducible graphs, *Discrete Applied Mathematics* **3**, (1981), pp. 163–174.
6. Corneil, D. G., Y. Perl, and L. K. Stewart, A linear recognition algorithm for cographs, *SIAM Journal on Computing* **14**, (1985), pp. 926–934.
7. Cornuéjols, Gérard, Xinming Liu, and Kristina Vušković, A polynomial algorithm for recognizing perfect graphs, *Proceedings of the* 44^{th} *Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, 2003.
8. Golumbic, M. C., *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
9. Golumbic, M. C. and M. Lipshsteyn, Chordal probe graphs (extended abstract), *Proceedings WG'2003*, LNCS 2880, (2003), pp. 249–260.
10. Golumbic, M. C. and Ann N. Trenk, *Tolerance Graphs*, Cambridge studies in advanced mathematics 89, 2004.
11. Grötschel, Martin, Characterizations of perfect graphs, *Mathematical Programming Society Newsletter* **62**, (1999).
12. Habib. M. and C. Paul, A simple linear time algorithm for cograph recognition, *Discrete Applied Mathematics* **145**, (2005), pp. 183–197.
13. Hayward, R. B., Weakly triangulated graphs, *Journal of Combinatorial Theory, series B* **39**, (1985), pp. 200–208.
14. Hoàng, C. T., *Perfect graphs*, PhD thesis, School of Computer Science, McGill University, Montreal 1985.
15. Jamison, B. and S. Olariu, P_4-reducible graphs—a class of uniquely tree representable graphs, *Studies in Appl. Math.* **81**, (1989), pp. 79–87.
16. Jamison, B. and S. Olariu, A unique tree representation for P_4-sparse graphs, *Discrete Applied Mathematics* **35**, (1992), pp. 115–129.
17. Jamison, B. and S. Olariu, Recognizing P_4-sparse graphs in linear time, *SIAM Journal on Computing* **21**, (1992), pp. 381–406.
18. Johnson, J. L. and J. Spinrad, A polynomial-time recognition algorithm for probe interval graphs, *Proceedings* 12^{th} *ACM–SIAM Symposium on Discrete Algorithms* (2001), pp. 477–486.
19. Lovász, L., A characterization of perfect graphs, *Journal of Combinatorial Theory Series B* **13**, (1972), pp. 95–98.
20. McConnell, R. M. and J. Spinrad, Construction of probe interval graphs, *Proceedings* 13^{th} *ACM–SIAM Symposium on Discrete Algorithms* (2002), pp. 866–875.
21. McMorris, F.R., Chi Wang, and P. Zhang, On probe interval graphs, *Discrete Applied Mathematics* **88**, (1998), pp. 315–324.
22. Tsukiyama, S., M. Ide, H. Ariyoshi, and I. Shirakawa, A new algorithm for generating all the maximal independent sets, *SIAM Journal on Computing* **6**, (1977), pp. 505–517.
23. Zhang, P, E. A. Schon, S. G. Fisher, E. Cayanis, J. Weiss, S. Kistler, and P. E. Bourne, An algorithm based on graph theory for the assembly of contigs in physical mapping of DNA, *CABIOS* **10**, (1994), pp. 309–317.

Power Domination Problem in Graphs

Chung-Shou Liao* and Der-Tsai Lee*

Dept. of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan
{shou794,dtlee}@iis.sinica.edu.tw

Abstract. To monitor an electric power system by placing as few phase measurement units (PMUs) as possible is closely related to the famous vertex cover problem and domination problem in graph theory. A set S is a power dominating set (PDS) of a graph $G = (V, E)$, if every vertex and every edge in the system is observed following the observation rules of power system monitoring. The minimum cardinality of a PDS of a graph G is the power domination number $\gamma_p(G)$. We show that the problem of finding the power domination number for split graphs, a subclass of chordal graphs, is NP-complete. In addition, we present a linear time algorithm for finding $\gamma_p(G)$ of an interval graph G, if the interval ordering of the graph is provided, and show that the algorithm with $O(n \log n)$ time complexity, is asymptotically optimal, if the interval ordering is not given, where n is the number of intervals. We also show that the same results hold for the class of proper circular-arc graphs.

1 Introduction

To continually monitor the power system and observe all the states, like: voltage magnitude at loads and the current phase measurement at branches [1] is an important task for electric power companies. Placing phase measurement units (PMUs) at selected bus locations in the power system is one of the efficient methods to monitor the power system. Because of their high cost, the number of PMUs has to be minimized while maintaining the ability to monitor and observe the system. A power system is said to be *observed* if all the states can be determined by a set of PMUs according to the following rules [1]:

1. Assign a state of current phase measurement to each branch incident to a bus provided with a PMU (also, assign a state of voltage measurement to each bus located with a PMU);
2. Assign a state of voltage measurement to every bus incident to a branch with known current and the other end bus with known voltages by using Ohm's law;
3. Assign a state of current phase measurement to each branch connecting two buses with with a known voltage by using Ohm's law;
4. Assign a state of current phase measurement to a branch whose current can be inferred by using Kirchhoff's current law

* Also with the Institute of Information Science, Academia Sinica, Nankang, Taipei 115, Taiwan

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 818–828, 2005.

Let $G = (V, E)$ be a graph representation of an electric power system, where a vertex represents an electric node (a substation bus where transmission branches, loads, and generators are connected) and an edge represents a transmission branch joining two electric nodes. The problem of locating a smallest set of PMUs to observe all the states of the power system is closely related to the famous vertex cover problem and domination problem. The power system observation problem can be transformed into the following graph-theoretic problem [5–7]. Given a graph $G = (V, E)$, a set $S \subseteq V$ is said to be a *power dominating set* (PDS) if every vertex and edge in G are observed by S according to the following observation rules corresponding to the above-mentioned PMU observation rules:

1. Any vertex a PMU is placed and its incident edges are observed.
2. If one end vertex of an observed edge is observed, then the other end vertex is observed.
3. Any edge connecting two observed vertices is observed.
4. If a vertex is of degree $k > 1$, and $k - 1$ of these incident edges are observed, then all k incident edges are observed.

The minimum cardinality of a PDS of a graph G is called the *power domination number* of G, denoted $\gamma_p(G)$. A set $D \subseteq V(G)$ is said to be a *dominating set* in a graph $G = (V, E)$ if every vertex in $V \setminus D$ is adjacent to at least a vertex in D. The cardinality of a minimum dominating set of a graph G is called the *domination number* of G, denoted $\gamma(G)$. A *vertex cover* of a graph $G = (V, E)$ is a set $C \subseteq V(G)$ such that C contains at least one end vertex of every edge in $E(G)$. The cardinality of a minimum vertex cover of a graph G is denoted $\beta(G)$. It is obvious that $1 \leq \gamma_p(G) \leq \gamma(G)$ and $1 \leq \gamma_p(G) \leq \beta(G)$ for any graph G. In [5], Haynes *et al.* considered the power domination problem as a variation of the domination problem and studied the relationship between them.

2 Notation and Definitions

A graph $H = (V_H, E_H)$ is a subgraph of $G = (V, E)$ if $V_H \subseteq V$ and $E_H \subseteq E$ and it is an *induced subgraph* of G if for all $u, v \in V_H$, $\overline{u, v} \in E_H$ if and only if $\overline{u, v} \in E$. If $V_H = \{v_i, \ldots, v_k\}$, the induced subgraph $H = (V_H, E_H)$ is also written as $\{v_i, \ldots, v_k\}_G$. The subscript G denoting the underlying graph will be omitted in the following without any confusion. A vertex $w \in V$ is said to be *a neighbor of* or *adjacent to* a vertex $v \in V$ if $\overline{v, w} \in E$. The *neighborhood* of a vertex $v \in V$ is $N_G(v) = \{w \in V : \overline{v, w} \in E\}$. The *closed neighborhood* of $v \in V$ is $N_G[v] = N_G(v) \cup \{v\}$. The *closed neighborhood* of a vertex set S, $N_G[S] = \bigcup_{s \in S} N_G[s]$. We define the *out-degree* of $v \in V_H$ of an induced subgraph $H = (V_H, E_H)$ of G to be the number of vertices in $V \setminus V_H$ adjacent to v, and the edge $\overline{v, w} \in E$ connecting a vertex $v \in V_H$ and $w \notin V_H$ is called an *out-going* edge. The observation rules for a vertex set $S = V^0$ where PMUs are placed can be rephrased as follows.

Induced Observation Rule

1. The sets of vertices and edges in the induced subgraph $K^1 = (V^1, E^1)$ of G are observed, where V^1 is the closed neighborhood of V^0. That is $V^1 = N_G[V^0]$.
2. The sets of vertices and edges in the induced subgraph $K^i = (V^i, E^i)$ of G are observed, where $V^i = V^{i-1} \cup \{w \mid \overline{v,w}$ is an out-going edge and $v \in V^{i-1}$ is of out-degree $1\}$, $i = 2, 3, 4, \ldots$.

Note that the new edge $\overline{v,w} \in E$, where $v \in V^{i-1}$ is of out-degree 1, defined in **Induced Observation Rule** 2, is exactly what is specified in the fourth observation rule. The final graph $K^i = K^{i-1}$ for some $i > 0$ is called the *observed graph* of V^0, denoted $\mathcal{G}_{V^0}$, and the *size* of $\mathcal{G}_{V^0}$, denoted $|\mathcal{G}_{V^0}|$ is defined to be the number of the vertices in V^i, i.e., $|\mathcal{G}_{V^0}| = |V^i|$. The set V^0 is a PDS of G if $\mathcal{G}_{V^0} = G$. The vertex set V^0 of the induced subgraph $K^0 = (V^0, E^0)$ of G is referred to as the *kernel*, and vertices in the kernel are referred to as the kernel vertices. The subsequent vertex sets V^i, $i > 0$ are *derived kernels* of the i^{th} generation. For ease of reference the vertices $V^i \setminus V^{i-1}$, $i > 0$ are called the i^{th} *generation descendants* or *i-descendants* for short, of those in V^0. Consider two kernels A and B and the observed graphs $\mathcal{G}_A$ and $\mathcal{G}_B$ respectively. Kernel A and kernel B are said to be *independent*, if $|\mathcal{G}_{A\cup B}|$ is equal to $|\mathcal{G}_A \cup \mathcal{G}_B|$. Otherwise, i.e., $|\mathcal{G}_{A\cup B}| > |\mathcal{G}_A \cup \mathcal{G}_B|$, they are *dependent*. The properties below follow from the **Induced Observation Rule**.

Property 1. For two vertex sets, U and W of a graph G, if $N_G[U] \subseteq N_G[W]$, then $\mathcal{G}_U \subseteq \mathcal{G}_W$. That is, the observed graph of kernel U is contained in the observed graph of kernel W.

Property 2. Given a graph $G = (V, E)$, two kernels $A, B \subseteq V$ and their observed graphs $\mathcal{G}_A$, $\mathcal{G}_B$, kernel A and kernel B are dependent, that is, $|\mathcal{G}_{A\cup B}| > |\mathcal{G}_A \cup \mathcal{G}_B|$ if and only if there is a vertex $v \in \mathcal{G}_A$ of out-degree k, such that among the k vertices adjacent to v, $k-1$ of them are in $\mathcal{G}_B$, or vice versa.

3 NP-Completeness Results

In this section we establish NP-completeness results for the power domination problem for split graphs (a subclass of chordal graphs). Given a graph $G = (V, E)$, a *stable set* is a subset of V in which all vertices are pairwise non-adjacent, and a *clique* is a subset of V in which all vertices are pairwise adjacent. A graph is a *bipartite* graph, if its vertex set can be partitioned into two stable sets. A graph is a *split* graph, if its vertex set can be partitioned into a stable set and a clique. Note that a split graph is *chordal*, that is, for every cycle of length greater than three, there must be an edge, called *chord*, connecting two non-consecutive vertices in the cycle. In [5] Haynes *et al.* gave NP-completeness proofs of the power domination problem for bipartite and chordal graphs. However, there is a flaw in the NP-completeness proof for chordal graphs. In their reduction proof from the well-known NP-complete problem 3SAT to the power domination problem for chordal graphs, they constructed an instance of *chordal* graph $G(C)$ from an instance C of 3SAT as follows. For each variable u_i, construct a complete

subgraph of four vertices, two of which are labeled u_i and $\bar{u_i}$. For each clause $C_j = \{u_i, u_k, u_l\}$, create two nonadjacent vertices labeled $C_{j,1}$ and $C_{j,2}$, and add edges: $u_i, C_{j,1}$, $u_i, C_{j,2}$, $u_k, C_{j,1}$, $u_k, C_{j,2}$, $u_l, C_{j,1}$, $u_l, C_{j,2}$. However, the graph $G(C)$ so constructed is not always a chordal graph.

Now we establish the NP-hardness of the power domination problem for split graphs by transforming to it from the vertex cover problem, which is known to be NP-complete. The **vertex cover decision problem** is defined as follows. Given a nontrivial graph and a positive integer k, determine if there is a vertex set of size at most k such that each edge of the graph has at least one end vertex in this set. The **power domination decision problem** is defined as follows. Given a graph and a positive integer k', determine if there exists a vertex set of size at most k' such that every vertex and every edge are observed by this set according to the observation rules. Similar NP-complete proofs are given in [6, 7, 10]. Due to space limitation, we give without proofs the following theorem. For details the reader is referred to [11].

Theorem 1. *The power domination decision problem is NP-complete for split graphs.*

4 Power Dominating Set for Interval Graphs

A graph G is called an *interval graph* if its vertices can be put into a one-to-one correspondence with a set of intervals I on the real line such that two vertices are connected by an edge of G if and only if their corresponding intervals have nonempty intersection. We call I an interval representation for G. It is known[4] that the class of interval graphs is a subclass of chordal graphs. Interval graphs were studied extensively for the domination problem, see, e.g., [3, 4, 6, 7]. In particular, most of the variations of the domination problem are solvable for this class of graphs. We shall assume in what follows that an interval representation of the interval graph is available. That is, suppose $G = (V, E)$ is an interval graph, and its interval representation $\{I_i = [a_i, b_i] : 1 \leq i \leq n\}$, is indexed so that their right endpoints satisfy the property: $b_1 \leq b_2 \leq ... \leq b_n$, and this ordering satisfies the following interval ordering (IO) property [13]:

Property 3. $G = (V, E)$ is an interval graph if and only if there exists an interval ordering $v_1, v_2, \ldots, v_n$ such that the following condition holds.

(IO) If $i < j < k$ and $\overline{v_i, v_k} \in E$, then $\overline{v_j, v_k} \in E$.

In this section, we present a linear time algorithm for the power domination problem for interval graphs if an interval ordering of the given interval graph is provided. In the following, we assume all the graphs we discussed are connected. First, we introduce the concept of *gap* for choosing PMUs. Given an interval graph $G = (V, E)$ with interval ordering $\{v_1, v_2, \ldots, v_n\}$, i.e., the corresponding intervals $I_i = [a_i, b_i]$, $\forall i$, with $b_1 \leq b_2 \leq \ldots \leq b_n$, we assume without loss of generality, that the left endpoint ordering of all intervals is also provided, that is, $a'_1 \leq a'_2 \leq \ldots \leq a'_n$. For two successive right endpoints b_i and b_{i+1}, $i \geq 1$,

the pair (b_i, b_{i+1}) is called a *b-gap* if there exists no vertex $v_k \neq v_{i+1}$ whose left endpoint a_k satisfies $b_i < a_k \leq b_{i+1}$. Similarly, an *a-gap* is a pair (a'_i, a'_{i+1}) of two successive left endpoints such that there exists no vertex $v_k \neq v'_i$ whose right endpoint b_k satisfies $a'_i \leq b_k < a'_{i+1}$. A *b*- or *a*-gap may contain more than two successive right or left, respectively, endpoints. Thus for each *b*-gap we define the first and last right endpoints, i.e., b_{f_i} and b_{ℓ_i}, respectively, and similarly for each *a*-gap we define the first and last left endpoints, i.e., a'_{f_i} and a'_{ℓ_i}, respectively. The set of all the endpoints on the real line can be marked with a sequence of labels, *a* and *b* representing left and right endpoints, respectively. When consecutive *a*'s are grouped collectively, we ignore the singleton *b*'s that are scattered in between joining their preceding *a*'s that define some interval, and mark the *a*-gap, $\mathcal{A}$. Similarly, when consecutive *b* 's are grouped collectively, we ignore the singleton *a*'s scattered in between joining their succeeding *b*'s that define some interval, and mark the *b*-gap, $\mathcal{B}$. Thus, we shall obtain a sequence of $\mathcal{A}$'s and $\mathcal{B}$'s interleaved with *a*'s and *b*'s. Note that there may be overlap of successive $\mathcal{A}$ and $\mathcal{B}$. For instance, given the subsequence $a'_i, b_j, a'_{i+1}, b_{j+1}$, where a'_i and b_j define interval v_j and a'_{i+1} and b_{j+1} define interval v_{j+1}. Then by our definition, a'_i, b_j, a'_{i+1} form an *a*-gap and b_j, a'_{i+1}, b_{j+1} form a *b*-gap. Similarly, there also may be overlap of successive $\mathcal{B}$ and $\mathcal{A}$. By pre-processing, we find all the *a*- and *b*-gaps, $ag_1, ag_2, \ldots, ag_p$ and $bg_1, bg_2, \ldots, bg_r$, respectively, where $ag_i = (a'_{f_i}, a'_{\ell_i})$ and $bg_j = (b_{f_j}, b_{\ell_j})$, to form the above sequence of $\mathcal{A}$'s and $\mathcal{B}$'s interleaved with *a*'s and *b*'s. In addition, the size of *a*-gap ag_i (*b*-gap bg_j), denoted $|ag_i|$ ($|bg_j|$), is defined to be the number of vertices defining this *a*-gap (*b*-gap), i.e., $|\{v'_{f_i}, v'_{f_i+1}, \ldots, v'_{\ell_i-1}, v'_{\ell_i}\}|$ ($|\{v_{f_j}, v_{f_j+1}, \ldots, v_{\ell_j-1}, v_{\ell_j}\}|$). Obviously, $|ag_i|, |bg_j| \geq 2$, for each $1 \leq i \leq p$, $1 \leq j \leq r$. The notion of gaps plays an important role in our algorithm, as it will become clear later.

Lemma 1. (blocking gap lemma) *The b-gap $bg_i = (b_{f_i}, b_{\ell_i})$, $1 \leq i \leq r$ is a blocking gap of any vertex v_j where the left endpoint of the interval corresponding to v_j lies to the right of b_{ℓ_i}, i.e., $a_j > b_{\ell_i}$. That is, each vertex v_u with $b_u \leq b_{\ell_i}$ does not belong to the observed graph of $\{v_j\}$. The a-gap $ag_i = (a'_{f_i}, a'_{\ell_i})$, $1 \leq i \leq p$, is a blocking gap of any vertex v_k where the right endpoint of the interval corresponding to v_k lies to the left of a'_{f_i}, i.e., $b_k < a'_{f_i}$. That is, each vertex v_u with $a_u \geq a'_{f_i}$ does not belong to the observed graph of $\{v_k\}$.*

For ease of reference we say that the *b*-gap $bg_i = (b_{f_i}, b_{\ell_i})$, $1 \leq i \leq r$ is a *left* blocking gap of any vertex v_j that lies to the *right* of b_{ℓ_i}, i.e., $a_j > b_{\ell_i}$, and that the *a*-gap $ag_i = (a'_{f_i}, a'_{\ell_i})$, $1 \leq i \leq p$, is a *right* blocking gap of any vertex v_k that lies to the *left* of a'_{f_i}, i.e., $b_k < a'_{f_i}$.

For any vertex v_j, among all left blocking *b*-gaps, $bg_i = (b_{f_i}, b_{\ell_i})$, for some i, $1 \leq i \leq r$, the one with the largest b_{ℓ_i} that is less than a_j, is referred to as **the** left blocking *b*-gap of v_j. Similarly, for any vertex v_k, among all right blocking *a*-gaps, $ag_i = (a'_{f_i}, a'_{\ell_i})$, for some i, $1 \leq i \leq p$, the one with the smallest a'_{f_i}, that is greater than b_k, is referred to as **the** right blocking *a*-gap of v_k.

Associated with a *b*-gap $bg_i = (b_{f_i}, b_{\ell_i})$, $1 \leq i \leq r$, we have a *PMU candidate* or *candidate* for short, v_{c_i}, which is the vertex adjacent to v_{f_i} and whose corresponding interval has the maximum right endpoint b_{c_i}, among those with this

property. Recall that vertex v_{c_i} corresponds to an interval $[a_{c_i}, b_{c_i}]$. Note that all the vertices $v_{f_i}, v_{f_i+1}, \ldots, v_{\ell_i-1}, v_{\ell_i}$ defining bg_i are in $N[v_{c_i}]$. Assume that the *PMU candidate* v_{c_r} associated with the last b-gap $bg_r = (b_{f_r}, b_{\ell_r})$, where $b_{\ell_r} = b_n$, is v_n. It is trivial that the vertices $v_{f_i}, v_{f_i+1}, \ldots, v_{\ell_i-1}, v_{\ell_i}$ that define the b-gap $bg_i = (b_{f_i}, b_{\ell_i})$, $1 \leq i \leq r$, are all contained in $\mathcal{G}_{\{v_{c_i}\}}$.

Let the b-gap $bg_k = (b_{f_k}, b_{\ell_k})$ be the *left* blocking gap of v_{c_i}, and the a-gap $ag_j = (a'_{f_j}, a'_{\ell_j})$ be the *right* blocking gap of v_{c_i}. The following lemmas follow immediately.

Lemma 2. (backward observation lemma) *The induced subgraph* $\{v_{\ell_k+1}, \ldots, v_{f_i}, \ldots, v_{\ell_i}, \ldots, v_{c_i}\}$ *is contained in the observed graph* $\mathcal{G}_{\{v_{c_i}\}}$ *of the kernel* $\{v_{c_i}\}$. *We call this generation of observed vertices and edges a backward observation from the kernel* $\{v_{c_i}\}$. *The backward observation from* $\{v_{c_i}\}$ *stops at the left blocking gap* bg_k *of* v_{c_i}.

Lemma 3. (forward observation lemma) *Let* v'_u *be the vertex to the immediate right of* a_{c_i}, *i.e.,* $a'_u > a_{c_i}$ *and between them there exists no other left endpoint. The induced subgraph* $\{v_{c_i}, v'_u, v'_{u+1}, \ldots, v'_{f_j-1}\}$ *is contained in the observed graph* $\mathcal{G}_{\{v_{c_i}\}}$ *of the kernel* $\{v_{c_i}\}$. *We call this generation of observed vertices and edges a forward observation from the kernel* $\{v_{c_i}\}$. *The forward observation from* $\{v_{c_i}\}$ *stops at the right blocking gap* ag_j *of* v_{c_i}.

The following lemma illustrates the role of PMU's on which we base our algorithm.

Lemma 4. *Given an interval graph* $G = (V, E)$, *there exists an optimal PDS* S *for* G *consisting exclusively of PMU candidates associated with* b*-gaps.*

We introduce our main idea to solve the power domination problem in connected interval graphs. Initially, the PMU candidate v_{c_1} associated with the first b-gap bg_1 has to be chosen because of Lemmas 1 and 4. If we chose $v_{c_j} \neq v_{c_1}$ with $j > 1$, then bg_1 would be the left blocking gap of v_{c_j} and v_1, in particular, would not belong to $\mathcal{G}_{\{v_{c_j}\}}$. Then the forward observation from $\{v_{c_1}\}$ will proceed until the right blocking gap of v_{c_1}. We consider the choice of the next PMU candidate v_{c_i} in a greedy manner such that all the vertices between v_{c_1} and v_{c_i} belong to $\mathcal{G}_{\{v_{c_1}, v_{c_i}\}}$ and the index i is as large as possible. If we can choose the next candidate v_{c_i} correctly, then by repeating the same strategy we claim that we will find the optimal PDS. To do this we need to consider the necessary and sufficient conditions that the kernels $\{v_{c_i}\}$ and $\{v_{c_k}\}$ are *complete*. We say that two different kernels $\{v_{c_i}\}$ and $\{v_{c_k}\}, c_i < c_k$ are *complete* if all the vertices between v_{c_i} and v_{c_k} belong to $\mathcal{G}_{\{v_{c_i}, v_{c_k}\}}$. Otherwise, they are said to be *incomplete*. Besides, we call that $\{v_{c_k}\}$ is *maximally complete* with respect to $\{v_{c_i}\}$ if they are *complete* and $c_k - c_i$ is maximum, i.e., we cannot find a vertex $c_j > c_k$ such that $\{v_{c_i}\}$ and $\{v_{c_j}\}$ are complete.

First, we have some definitions. The *essential spot* of an a-gap $ag_i = (a'_{f_i}, a'_{\ell_i})$, denoted $ess(ag_i)$, is defined to be the second smallest right endpoint of the vertices $v'_{f_i}, v'_{f_i+1}, \ldots, v'_{\ell_i}$ defining the a-gap. In addition, we say a vertex set S

breaks a b-gap $bg_j = (b_{f_j}, b_{\ell_j})$ (respectively, an a-gap $ag_i = (a'_{f_i}, a'_{\ell_i})$) if at least $|bg_j| - 1$ vertices among $v_{f_j}, \ldots, v_{\ell_j}$ (respectively, at least $|ag_i| - 1$ vertices among $v'_{f_j}, \ldots, v'_{\ell_j}$) belong to the observed graph $\mathcal{G}_S$. Obviously, if the *essential spot*, $ess(ag_i)$, of some a-gap ag_i lies to the immediate right of some b-gap bg_k, i.e., $ess(ag_i) > b_{\ell_k}$, then the kernel $\{v_{c_{k+1}}\}$ *breaks* this a-gap ag_i by Property 3 and Lemma 2.

We use the next procedure to introduce the formal definition of *alternating break* of the blocking gaps between two kernels $\{v_{c_i}\}$ and $\{v_{c_k}\}$. First, for convenience, we define that the *essential spot* of a b-gap $bg_i = (b_{f_i}, b_{\ell_i})$, denoted $ess(bg_i)$, is the second largest left endpoint of the vertices $v_{f_i}, v_{f_{i+1}}, \ldots, v_{\ell_i}$ defining the b-gap (similar to the previous definition of essential spot of a-gap). Besides, we use two *linked lists* $A[\,]$ and $B[\,]$ to store the a-gap and b-gap sequences, respectively. For each ag (bg, respectively), we add the pointer from $ess(ag)$ to its immediate right b-gap (from $ess(bg)$ to its immediate left a-gap, respectively). The procedure **Alternate Break** is as follows.

Procedure Alternate Break(v_{c_i}, v_{c_k}, $A[\,]$, $B[\,]$)

1. Let $j = 1$ and $temp = 0$;
2. **while**(there is a pair of gaps, a left a-gap and a right b-gap between v_{c_i}, v_{c_k})
   { Let the leftmost a-gap be ag_{i_j} and the rightmost b-gap be bg_{k_j};
   if(the backward observation from v_{c_k} breaks ag_{i_j})
   { Delete the broken a-gap, update $A[\,]$ and the corresponding $ess(bg)$;
   $temp = temp + 1$; }
   if(the forward observation from v_{c_i} breaks bg_{k_j})
   { Delete the broken b-gap, update $B[\,]$ and the corresponding $ess(ag)$;
   $temp = temp + 1$; }
   if($temp \neq 2$)
   { **Return Failure**; }
   $j = j + 1$ and let $temp = 0$; }
3. **Return Success**;

The above procedure **Alternate Break** shows the repeated process of the propagation of the forward observation from $\{v_{c_i}\}$ and the backward observation from $\{v_{c_k}\}$ alternately. We call this process an *alternating break* by two kernels $\{v_{c_i}\}$ and $\{v_{c_k}\}$. By this definition, we characterize the notion of completeness in the following lemma.

Lemma 5. *Given a connected interval graph $G = (V, E)$, kernels $\{v_{c_i}\}$ and $\{v_{c_k}\}$, $c_i < c_k$, are complete, i.e., all the vertices between v_{c_i} and v_{c_k} belong to the observed graph $\mathcal{G}_{\{v_{c_i}, v_{c_k}\}}$ if and only if between v_{c_i} and v_{c_k} there exists neither a-gap nor b-gap that remains to be unbroken via alternating break by two kernels $\{v_{c_i}\}$ and $\{v_{c_k}\}$, that is, the procedure* **Alternate Break** *returns Success.*

Corollary 1. *Given a connected interval graph $G = (V, E)$, if the different kernels $\{v_{c_i}\}$ and $\{v_{c_k}\}$, $c_i < c_k$, are incomplete, then there must exist at least one a-gap and one b-gap between v_{c_i} and v_{c_k}, which are unbroken via alternating break by the kernels $\{v_{c_i}\}$ and $\{v_{c_k}\}$.*

Based on the above, we present a linear time algorithm MPDI for solving the power domination problem in a given connected interval graph. The correctness and timing analysis of the algorithm can be found in [11].

Algorithm MPDI. Find a minimum PDS of a connected interval graph.
Input. A connected interval graph $G = (V, E)$ with interval ordering $v_1, v_2, \ldots, v_n$. A linked list $A[\,]$ consists of all the a-gaps $ag_1, ag_2, \ldots, ag_p$ and their essential spots $ess(ag_i)$, $1 \leq i \leq p$, and also a linked list $B[\,]$ consists of all the b-gaps $bg_1, bg_2, \ldots, bg_r$ and their PMU candidates $v_{c_1}, v_{c_2}, \ldots, v_{c_r}$.
Output. A minimum PDS S of G.
Method.

1. Let $S = \{v_{c_1}\}$ and $v_c = v_{c_1}$;
2. Find the right blocking a-gap ag_i of v_c;
3. Delete the b-gaps, if any, broken by the forward observation from the kernel $\{v_c\}$ and update $B[\,]$ and corresponding $ess(ag)$ pointers;
4. Select a possible candidate $v_c^* = v_{c_k}$ associated with b-gap bg_k where $ess(ag_i)$ either lies to the immediate left of or belongs to bg_k;
5. /* Check completeness conditions for $\{v_c\}$ and $\{v_c^*\}$. */
   while(**Alternate Break**(v_c, v_c^*, $A[\,]$, $B[\,]$) returns **Failure**)
   { Let the first a-gap blocking the forward observation be ag_{i_j};
   Select the new possible candidate $v_c^* = v_{c_{k'}}$ associated with b-gap $bg_{k'}$, where $ess(ag_{i_j})$ lies to the immediate left of or belongs to $bg_{k'}$; }
6. Put the maximally complete candidate v_c^* w.r.t. the kernel v_c into S;
7. Let $v_c = v_c^*$; repeat Steps 2 to 7 until there is no right blocking a-gap of v_c.

Theorem 2. *Given a connected interval graph $G = (V, E)$, Algorithm MPDI produces an optimal PDS S of minimum cardinality for G.*

Theorem 3. *Algorithm MPDI takes $\Theta(nlogn)$ time, which is asymptotically optimal. In addition, it takes $O(n)$ time provided that the endpoints of the intervalts are given sorted.*

5 Power Dominating Set for Proper Circular-Arc Graphs

We refer to [8], [12], [14] and consider the power domination problem in circular-arc graphs. A graph G is called a *circular-arc graph* if its vertices can be put into a one-to-one correspondence with a set of arcs on a circle such that two vertices are adjacent in G if and only if their correspondence arcs have nonempty intersection. We call this set of arcs on a circle a circular-arc representation. A graph G is said to be *proper* if for the corresponding intervals or arcs of vertex set of G, no one is contained in another. For example, G is a *proper circular-arc graph* if no arc is contained in another arc in G. A circular-arc is denoted (a_i, b_i), where b_i follows a_i in clockwise direction and a_i and b_i are called the left and right endpoints respectively. Note that arc (b_i, a_i) denotes the complement of

arc (a_i, b_i) with respect to the circle. We arbitrarily select a right endpoint and label it b_1, and proceed to label the subsequent right endpoints following b_1 in clockwise direction as $b_2, b_3, \ldots, b_n$, and the corresponding vertices have thus a circular ordering $v_1, v_2, \ldots, v_n$ with $b_1 \preceq b_2 \preceq \ldots b_n \preceq b_1$, where $b_i \preceq b_j$ means that b_j follows b_i in clockwise direction. Vertices between v_i and v_k refer to those vertices whose corresponding right endpoints b_j lie on the arc defined by (b_i, b_k), i.e., $b_i \preceq b_j \preceq b_k$. Two different kernels $\{v_{c_i}\}$ and $\{v_{c_k}\}$, $c_i \preceq c_k$, are said to be complete, if all the vertices between v_{c_i} and v_{c_k} belong to $\mathcal{G}_{\{v_{c_i}, v_{c_k}\}}$. $\{v_{c_k}\}$ is said to be maximally complete with respect to $\{v_{c_i}\}$ if they are complete and there exists no other $v_{c_{k'}}$ such that v_{c_k} lies between v_{c_i} and $v_{c_{k'}}$, and $\{v_{c_i}\}$ and $\{v_{c_{k'}}\}$ are complete.

All lemmas in Section 4 also hold for proper circular-arc graphs. We present a linear time algorithm for the power domination problem for proper circular-arc graphs, where no arc is contained in another arc, provided that a circular ordering of the proper circular-arc graph is given. We apply Step 2 to Step 5 of Algorithm MPDI to find the candidate $v_c^* = v_{c_k}$ such that $\{v_c^*\}$ is maximally complete with respect to $\{v_c = v_{c_i}\}$, $c_i \preceq c_k$. It is obvious that this process works in circular-arc graphs. We denote this candidate v_c^* as $\mathbf{NEXT}(v_c)$ if $\{v_c^*\}$ is maximally complete with respect to $\{v_c\}$. For the candidate v_c, if there exists no a-gap that is the right (or clockwise) blocking a-gap of v_c, then all the vertices following v_c clockwise belong to $\mathcal{G}_{\{v_c\}}$ and we assume $\mathbf{NEXT}(v_c)$= *null*. Then we have the following lemma to illustrate *interleaving property* of the v_c and $\mathbf{NEXT}(v_c)$ relation.

Lemma 6. (Interleaving Property) *Given a circular-arc graph G with a circular ordering $v_1, v_2, \ldots, v_n$, for any two distinct PMU candidates v_{c_i} and v_{c_j}, $c_i \preceq c_j$, we have $\mathbf{NEXT}(v_{c_i}) \preceq \mathbf{NEXT}(v_{c_j})$.*

We construct the directed graph $D = (V_D, E_D)$, where $V_D = \{v_{c_1}, v_{c_2}, \ldots, v_{c_r}\}$ and the directed edge $(v_{c_i}, v_{c_j}) \in E_D$ if and only if $v_{c_j} = \mathbf{NEXT}(v_{c_i})$. By assumption, G is a connected circular-arc graph with a circular ordering $v_1, \ldots, v_n$. First, we assume $V_D \neq \emptyset$, that is, at least one b-gap in G, else we let $\{v_n\}$ be a PDS. Next, we assume that every vertex $v_{c_i} \in V_D$ has its $\mathbf{NEXT}(v_{c_i})$, that is, $\mathbf{NEXT}(v_{c_i}) \neq$ *null*, $\forall i$; otherwise, if there exists some v_{c_i} with $\mathbf{NEXT}(v_{c_i})$= *null*, we select it as a PDS. Consequently, there exists at least one directed cycle in D because V_D is of finite cardinality. Besides, no two directed cycles can share one common vertex since every vertex has exactly one out-degree in D. We define $PDS(v_{c_i}) = \{v_{c_i}, v_{c_i}^{(1)}, \ldots, v_{c_i}^{(m)}\}$, where $v_{c_i}^{(j+1)} = \mathbf{NEXT}(v_{c_i}^{(j)})$, $v_{c_i}^{(0)} = v_{c_i}$, and either $v_{c_i} = \mathbf{NEXT}(v_{c_i}^{(m)})$ or $v_{c_i} \preceq \mathbf{NEXT}(v_{c_i}^{(m)})$. Undoubtedly, $PDS(v_{c_i})$ is a PDS containing v_{c_i} for a circular-arc graph G by definition. We have the following lemma.

Lemma 7. *Let S be any PDS of a circular-arc graph G and $v_{c_i} \in S$, for some i. Then we have $|PDS(v_{c_i})| \leq |S|$.*

By Lemma 7, we know that $PDS(v_{c_i})$ is the minimum PDS containing v_{c_i}. In addition, a vertex v_{c_i} in V_D is called a *valid candidate* if $|PDS(v_{c_i})| = \gamma_p(G)$ and we have the next result.

Lemma 8. *There is at least one directed cycle consisting exclusively of valid candidates in D.*

Theorem 4. *Every directed cycle C in the directed graph D consists exclusively of valid candidates.*

Note that the selection of candidate $v_c^* = v_{c_k}$ associated with the b-gap bg_k in Step 4 of Algorithm MPDI is exactly the one that is maximally complete with respect to $\{v_c\}$ in proper circular-arc graphs. The reason is that if $v_c^* = v_{c_k}$ breaks the a-gap ag_i, then it also breaks all the a-gaps between ag_i and bg_k since their essential spots follow $ess(ag_i)$ clockwise. Step 5 of Algorithm MPDI, that checks completeness by alternating break procedure, is unnecessary for proper circular-arc graphs. Based on the above, we have Algorithm MPDPC to solve the power domination problem in proper circular-arc graphs.

Algorithm MPDPC. Find a minimum PDS of a connected proper circular-arc graph.
Input. A connected proper circular-arc graph $G = (V, E)$ with circular ordering $v_1, v_2, \ldots, v_n$. All the a-gaps $ag_1, ag_2, \ldots, ag_p$ and their essential spots $ess(ag_i)$, $1 \leq i \leq p$, and also all the b-gaps $bg_1, bg_2, \ldots, bg_r$ and their PMU candidates $v_{c_1}, v_{c_2}, \ldots, v_{c_r}$.
Output. A minimum PDS S of G.
Method.

1. **if**(there is no b-gap) { Let $\{v_n\}$ be a PDS and **stop**; }
2. Let all PMU candidates be labeled "unvisited" and let $v_c = v_{c_1}$;
3. **while**(v_c is "unvisited")
   { Label v_c as "visited" and run Step 2 in MPDI;
   if(there is no clockwise or right blocking a-gap of v_c)
   { Let $\{v_c\}$ be a PDS and **stop**; }
   Run Steps 3 and 4 in MPDI to find v_c^* =**NEXT**(v_c);
   Let $v_c = v_c^*$; }
4. Select the vertex set of the $PDS(v_c)$ from v_c as a PDS;

Theorem 5. *Given a proper circular-arc graph $G = (V, E)$, Algorithm MPDPC produces an optimal PDS of minimum cardinality for G in linear time if the circular-arc endpoints are sorted.*

6 Concluding Remarks

We have considered the power domination problem, which is related to the domination problem in graph theory [5], and shown that the power domination problem for split graphs (a subclass of chordal graphs) in NP-complete. We have also presented a linear time algorithm for solving the power domination problem for both interval graphs and proper circular-arc problem, provided that the endpoints of the corresponding interval model or circular-arc model have been given

sorted. We conjecture that the power domination problem for general circular-arc graphs can be solved in linear time. There are several open problems: what are the complexities of the power domination problem for other classes of intersection graphs and what are the relationships, if any, between the power domination number and other variations of domination numbers.

References

1. T. L. Baldwin, L. Mili, M. B. Boisen, Jr., and R. Adapa, Power system observability with minimal phasor measurement placewement, *IEEE Trans. Power Syst.* Vol. 8, No. 2, May 1993, 707-715.
2. M. Ben-Or, Lower bounds for algebraic computation trees, *Proc. 15th Annual Symp. on theory of Computing*, 1983, pp. 80-86.
3. G. J. Chang, Algorithmic aspects of domination in graphs, in: *Handbook of Combinatorial Optimization* (D.-Z. Du and P. M. Pardalos eds.) Vol. 3 (1998) 339-405.
4. M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, Inc (1980).
5. T. W. Haynes, S. M. Hedetniemi, S. T. Hedetniemi, and M. A. Henning, Domination in graphs applied to electric power networks, *SIAM J. Discrete Math.* Vol. 15, No. 4, (2002) 519-529.
6. T. W. Haynes, S. T. Hedetniemi and P. J. Slater, *Domination in Graphs: The Theory*, Marcel Dekker, Inc. New York (1998).
7. T. W. Haynes, S. T. Hedetniemi and P. J. Slater, *Domination in Graphs: Advanced Topics*, Marcel Dekker, Inc. New York (1998).
8. Wen-Lian Hsu, and Kuo-Hui Tsai, Linear time algorithms on circular-arc graphs, *Infrom. Process. Letter.*, Vol. 40, (1991) 123-129.
9. C. S. Liao and G. J. Chang, Algorithmic aspect of k-tuple domination in graphs, *Taiwanese J. Math.* 6 (2002) 415-420.
10. C. S. Liao and G. J. Chang, k-tuple domination in graphs, *Inform. Process. Letter.*, Vol. 87, (2003) 45-50.
11. C. S. Liao and D. T. Lee, Power domination problem in graphs, manuscript, Institute of Information Science, Academia Sinica, April 2005.
12. D. T. Lee, M. Sarrafzadeh, and Y. F. Wu, Minimum cuts for circular-arc graphs, *SIAM J. Comput.*, Vol. 19, No. 6, (1990) 1041-1050.
13. G. Ramalingam and C. Pandu Rangan, A unified approach to domination problems in interval graphs, *Inform. Process. Letter.*, Vol. 27, (1988) 271-274.
14. K. H. Tsai anad D. T. Lee, k-Best Cuts for Circular-Arc Graphs, *Algorithmica*, Vol. 18, (1997) 198-216.
15. Min Zhao, Liying Kang, G. J. Chang, Power domination in graphs, manuscript, Personal Communication.

Complexity and Approximation of Satisfactory Partition Problems*

Cristina Bazgan[1], Zsolt Tuza[2], and Daniel Vanderpooten[1]

[1] LAMSADE, Université Paris-Dauphine, France
{bazgan,vdp}@lamsade.dauphine.fr
[2] Computer and Automation Institute, Hungarian Academy of Sciences, Budapest and Department of Computer Science, University of Veszprém, Hungary

Abstract. The Satisfactory Partition problem consists of deciding if a given graph has a partition of its vertex set into two nonempty parts such that each vertex has at least as many neighbors in its part as in the other part. This problem was introduced by Gerber and Kobler in 1998 and further studied by other authors but its complexity remained open until now. We prove in this paper that Satisfactory Partition, as well as a variant where the parts are required to be of the same cardinality, are *NP*-complete. We also study approximation results for the latter problem, showing that it has no polynomial-time approximation scheme, whereas a constant approximation can be obtained in polynomial time. Similar results hold for balanced partitions where each vertex is required to have *at most* as many neighbors in its part as in the other part.

1 Introduction

Gerber and Kobler introduced in [5, 6] the problem of deciding if a given graph has a vertex partition into two nonempty parts such that each vertex has at least as many neighbors in its part as in the other part. A graph with this property is called *satisfactory partitionable*. As remarked by Gerber and Kobler, Satisfactory Partition may have no solution. In particular, the following graphs are not satisfactory partitionable: complete graphs, stars, and complete bipartite graphs with at least one of the two vertex sets having odd size. Some other graphs are easily satisfactory partitionable: cycles of length at least 4, trees which are not stars, and disconnected graphs. After [5, 6] this problem was further studied in [8] and [1] but its complexity remained open until now, while some generalizations were studied and proved to be *NP*-complete.

We define in this paper another variant of Satisfactory Partition, called Balanced Satisfactory Partition, where the parts are required to have the same cardinality. A graph admitting such a partition is said to be *balanced satisfactory partitionable*. Graphs like cycles of even length and complete bipartite

* This research was supported by the bilateral research cooperation Balaton between EGIDE (France) and Ministry of Education (Hungary) under grant numbers 07244RJ and F-29/2003. The second author was also supported in part by the Hungarian Scientific Research Fund, grant OTKA T-042710

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 829–838, 2005.

graphs with both vertex classes of even size are trivially balanced satisfactory partitionable. A graph of even order formed by two non-partitionable connected components of unequal size, however, is an example of a graph which is satisfactory partitionable but not balanced satisfactory partitionable. We show in this paper that Satisfactory Partition and Balanced Satisfactory Partition are *NP*-complete.

We consider also the opposite problem of deciding if a given graph has a vertex partition into two parts such that each vertex has at least as many neighbors in the other part as in its own part. This problem called Co-Satisfactory Partition corresponds to finding in the graph a maximal cut with respect to moving a vertex from its part to the other. Therefore, a graph always admits such a partition that can be found in polynomial time. However, the balanced version of this problem, called Balanced Co-Satisfactory Partition, does not always admit a solution, e.g. for stars of even order. We prove in this paper that Balanced Co-Satisfactory Partition is *NP*-complete.

When a graph has no balanced (co-)satisfactory partition, it is natural to ask for a balanced partition maximizing the number of (co-)satisfied vertices. The corresponding optimization problems are Max Satisfying Balanced Partition and Max Co-Satisfying Balanced Partition. We prove in this paper that Max Satisfying Balanced Partition is 3-approximable, Max Co-Satisfying Balanced Partition is 2-approximable, and that these two problems have no polynomial-time approximation scheme unless *P=NP*.

The paper is structured as follows. Section 2 contains some notations and definitions of problems. In Section 3 we show the *NP*-completeness of Satisfactory Partition, Balanced Satisfactory Partition, and Balanced Co-Satisfactory Partition. In Section 4 we prove that Max (Co-)Satisfying Balanced Partition has no approximation scheme, unless *P=NP*, and in Section 5 we give constant approximation algorithms for these problems.

2 Preliminaries

We begin with some basic definitions concerning approximation, and then we define the problems considered.

Approximability. Given an instance x of an optimization problem A and a feasible solution y of x, we denote by $val(x, y)$ the value of solution y, and by $opt_A(x)$ the value of an optimum solution of x. For a function ρ, an algorithm is a *ρ-approximation* for a maximization problem A if for any instance x of the problem it returns a solution y such that $val(x, y) \geq \frac{opt_A(x)}{\rho(|x|)}$. We say that a maximization problem is *constant approximable* if, for some constant $\rho > 1$, there exists a polynomial-time ρ-approximation for it. A maximization problem has a *polynomial-time approximation scheme* (a PTAS, for short) if, for every constant $\varepsilon > 0$, there exists a polynomial-time $(1 + \varepsilon)$-approximation for it.

Reductions. ([7]) Let A and A' be two maximization problems. Then A is said to be *gap-preserving reducible* to A' with parameters $(c, \rho), (c', \rho')$ (where

$\rho, \rho' \geq 1$), if there is a polynomial-time algorithm that transforms any instance x of A to an instance x' of A' such that the following properties hold:

$$opt_A(x) \geq c \Rightarrow opt_B(x') \geq c' \qquad \text{and} \qquad opt_A(x) < \frac{c}{\rho} \Rightarrow opt_B(x') < \frac{c'}{\rho'}$$

Gap-preserving reductions have the following property. If it is NP-hard to decide if the optimum of an instance of A is at least c or less than $\frac{c}{\rho}$, then it is NP-hard to decide if the optimum of an instance of A' is at least c' or less than $\frac{c'}{\rho'}$. This NP-hardness implies that A' is hard to ρ'-approximate.

Graphs. We consider finite, undirected graphs without loops and multiple edges. For a graph $G = (V, E)$, a vertex $v \in V$, and a subset $Y \subseteq V$ we denote by $d_Y(v)$ the number of vertices in Y that are adjacent to v; and, as usual, we write $d(v)$ for the degree $d_V(v)$ of v in V. A partition (V_1, V_2) of V is said to be *nontrivial* if both V_1 and V_2 are nonempty.

The problems we are interested in are defined as follows.

Satisfactory Partition
Input: A graph $G = (V, E)$.
Question: Is there a nontrivial partition (V_1, V_2) of V such that for every $v \in V$, if $v \in V_i$ then $d_{V_i}(v) \geq d_{V_{3-i}}(v)$?

The variant of this problem where the two parts have equal size is:

Balanced Satisfactory Partition
Input: A graph $G = (V, E)$ on an even number of vertices.
Question: Is there a partition (V_1, V_2) of V such that $|V_1| = |V_2|$ and for every $v \in V$, if $v \in V_i$ then $d_{V_i}(v) \geq d_{V_{3-i}}(v)$?

Given a *partition* (V_1, V_2), we say that a vertex $v \in V_i$ is *satisfied* if $d_{V_i}(v) \geq d_{V_{3-i}}(v)$, or equivalently if $d_{V_i}(v) \geq \lceil \frac{d(v)}{2} \rceil$. A graph admitting a nontrivial partition where all vertices are satisfied is called *satisfactory partitionable*, and such a partition is called a *satisfactory partition*. If $|V_1| = |V_2|$ also holds, then it will be called a *balanced satisfactory partition* and the graph G is *balanced satisfactory partitionable*.

Co-Satisfactory Partition
Input: A graph $G = (V, E)$.
Question: Is there a partition (V_1, V_2) of V such that for every $v \in V$, if $v \in V_i$ then $d_{V_i}(v) \leq d_{V_{3-i}}(v)$?

We already mentioned in the introduction that Co-Satisfactory Partition always has a solution which can be found easily in polynomial time.

Balanced Co-Satisfactory Partition
Input: A graph $G = (V, E)$ on an even number of vertices.
Question: Is there a partition (V_1, V_2) of V such that $|V_1| = |V_2|$ and for every $v \in V$, if $v \in V_i$ then $d_{V_i}(v) \leq d_{V_{3-i}}(v)$?

Given a partition (V_1, V_2), a vertex $v \in V_i$ is *co-satisfied* if $d_{V_i}(v) \leq d_{V_{3-i}}(v)$, or equivalently if $d_{V_i}(v) \leq \lceil \frac{d(v)}{2} \rceil$. The previous notions are similarly defined for co-satisfiability.

When a graph is not balanced (co-)satisfactory partitionable, it is natural to ask for a balanced partition that maximizes the number of vertices that are (co-)satisfied. Therefore, we consider the following problems.

Max Satisfying Balanced Partition
Input: A graph $G = (V, E)$ on an even number of vertices.
Output: A partition (V_1, V_2) of V, such that $|V_1| = |V_2|$, that maximizes the number of satisfied vertices.

Max Co-Satisfying Balanced Partition
Input: A graph $G = (V, E)$ on an even number of vertices.
Output: A partition (V_1, V_2) of V, such that $|V_1| = |V_2|$, that maximizes the number of co-satisfied vertices.

Almost Balanced Partitions. The above problems can also be formulated for graphs with an odd number of vertices, requiring partitions (V_1, V_2) such that $|V_1|$ and $|V_2|$ differ just by 1.

3 Complexity of (Balanced) (Co-)Satisfactory Partition

In this section we establish the *NP*-completeness of the following three problems:
(i) Satisfactory Partition
(ii) Balanced Satisfactory Partition
(iii) Balanced Co-Satisfactory Partition
The overall scheme is that (iii) is *NP*-complete, (iii) is reducible to (ii), and (ii) is reducible to (i).

Proposition 1. Balanced Satisfactory Partition *is polynomial-time reducible to* Satisfactory Partition.

Proof. Let $G = (V, E)$ be a graph, instance of the first problem on n vertices. The graph $G' = (V', E')$, instance of Satisfactory Partition, is obtained from G by adding two cliques of size $\frac{n}{2}$, $A = \{a_1, \ldots, a_{\frac{n}{2}}\}$ and $B = \{b_1, \ldots, b_{\frac{n}{2}}\}$. In G', in addition to the edges of G, all vertices of V are adjacent to all vertices of A and B. Also each vertex $a_i \in A$ is linked to all vertices of B except b_i, $i = 1, \ldots, \frac{n}{2}$.

Let (V_1, V_2) be a balanced satisfactory partition of G. Then (V_1', V_2') where $V_1' = V_1 \cup A$ and $V_2' = V_2 \cup B$ is a satisfactory partition of G'. Indeed, a vertex from $A \cup B$ is satisfied, for example if $v \in A$, $d_{V_1'}(v) = |V_1| + |A| - 1 = |V_2| + |B| - 1 = d_{V_2'}(v)$. Also it is easy to see that a vertex from V is satisfied in G' since it is satisfied in G.

Let (V_1', V_2') be a satisfactory partition of G', where $V_1' = V_1 \cup A_1 \cup B_1$ and $V_2' = V_2 \cup A_2 \cup B_2$ with $V_i \subseteq V, A_i \subseteq A, B_i \subseteq B, i = 1, 2$. We claim that (V_1, V_2) is a balanced satisfactory partition of G.

We first show that $A_1 \cup B_1 \neq \emptyset$ and $A_2 \cup B_2 \neq \emptyset$, which means that no satisfactory partition can contain $A \cup B$ in one of its parts. Indeed, by contradiction, suppose we have $V_1' = V_1 \cup A \cup B$ and $V_2' = V_2$. Then, the inequality specifying that $v \in V_2$ is satisfied is $d_{V_2}(v) \geq d_{V_1}(v) + n$ which is impossible. So,

two cases are possible: either each part of the partition contains one clique, say $V_1' = V_1 \cup A$ and $V_2' = V_2 \cup B$ (case 1) or at least one of the cliques is cut by the partition (case 2).

In case 1, in order for a vertex of A to be satisfied, we have $\frac{n}{2} - 1 + |V_1| \geq |V_2| + \frac{n}{2} - 1$ and in order that a vertex of B be satisfied, we have $\frac{n}{2} - 1 + |V_2| \geq |V_1| + \frac{n}{2} - 1$. These two inequalities imply $|V_1| = |V_2|$. Moreover, since $v \in V_1 \cup V_2$ is satisfied in G' where it is linked to $\frac{n}{2}$ vertices in A and $\frac{n}{2}$ vertices in B, v is also satisfied in G.

In case 2, suppose that clique A is cut by the partition into non-empty sets A_1 and A_2 while B_1 or B_2 may be empty. We show now that if $a_i \in A_1$ for some i, then also $b_i \in B_2$ for the same i. Assume by contradiction that $b_i \in B_1$. Since a_i is satisfied we have

$$(|A_1| - 1) + (|B_1| - 1) + |V_1| \geq |A_2| + |B_2| + |V_2| \tag{1}$$

This implies $|V_1'| > |V_2'|$.

Let $a_j \in A_2$. We may have $b_j \in B_1$ or $b_j \in B_2$. If $b_j \in B_2$ then the condition that a_j is satisfied is

$$(|A_2| - 1) + (|B_2| - 1) + |V_2| \geq |A_1| + |B_1| + |V_1| \tag{2}$$

If $b_j \in B_1$ then the condition that a_j is satisfied is

$$(|A_2| - 1) + |B_2| + |V_2| \geq |A_1| + (|B_1| - 1) + |V_1| \tag{3}$$

Each of (2) and (3) implies that $|V_2'| \geq |V_1'|$, contradicting (1). Thus $|A_1| = |B_2|$ and $|A_2| = |B_1|$, that means that both cliques are cut by the partition.

For $a_i \in A_1$ and $b_i \in B_2$ the inequalities specifying that a_i and b_i are satisfied are respectively: $(|A_1| - 1) + |B_1| + |V_1| \geq |A_2| + (|B_2| - 1) + |V_2|$

and $|A_2| + (|B_2| - 1) + |V_2| \geq (|A_1| - 1) + |B_1| + |V_1|$

from which we obtain $|A_1| + |B_1| + |V_1| = |A_2| + |B_2| + |V_2|$. Since $|A_1| = |B_2|$ and $|A_2| = |B_1|$, we get $|V_1| = |V_2|$.

Moreover, since $v \in V_1 \cup V_2$ is satisfied in G' where it is linked to $|A_1| + |B_1| = \frac{n}{2}$ vertices in V_1' among the vertices of the two cliques and $|A_2| + |B_2| = \frac{n}{2}$ vertices in V_2', v is also satisfied in G. □

We state now our *NP*-completeness results.

Theorem 1. Balanced Co-Satisfactory Partition *is NP-complete.*

Proof. Clearly, this problem is in *NP*. We construct a polynomial reduction from a variant of Independent Set, the problem of deciding if a graph with n vertices contains an independent set of size at least $\frac{n}{2}$, a problem stated to be *NP*-hard in [4]. Let $G = (V, E)$ be a graph with n vertices $v_1, \ldots, v_n$ and m edges, an input of this variant of Independent Set problem. We assume that n is even, since otherwise we can add a vertex that we link with all the vertices of the graph without changing the problem. The edges of G are labelled $e_1, \ldots, e_m$. We construct a graph $G' = (V', E')$, instance of Balanced Co-Satisfactory Partition as follows: the vertex set V' consists of three sets F, T and V (the vertex set of G) where $F = \{f_1, \ldots, f_{2m+1}\}$ and $T = \{t_1, \ldots, t_{2m+1}\}$. Vertices $f_{2\ell}$, $f_{2\ell+1}$ correspond to edge e_ℓ ($\ell = 1, \ldots, m$) and f_1 is an additional vertex. F and T are two independent sets of size $2m + 1$. Vertices t_i are linked

with f_j, $i = 1, \ldots, 2m+1$, $j = 1, \ldots, 2m+1$. In addition to these edges and E, the edge set E' contains the edges $(f_{2\ell}, v_i)$ and $(f_{2\ell+1}, v_j)$ for each edge $e_\ell = (v_i, v_j)$, $\ell = 1, \ldots, m$.

It is easy to see that this construction can be accomplished in polynomial time. All that remains to show is that G has an independent set of size at least $\frac{n}{2}$ if and only if G' is balanced co-satisfactory partitionable.

Suppose firstly that G has an independent set of size at least $\frac{n}{2}$. Let S be an independent set of size exactly $\frac{n}{2}$ of G. Let $V_1' = F \cup S$ and $V_2' = T \cup \bar{S}$, where $\bar{S} = V \setminus S$. Let us check in the following that (V_1', V_2') is a balanced co-satisfactory partition. It is easy to see that all vertices of F and T are co-satisfied. Let $v \in S$. Since S is an independent set, v is not linked to any vertex in S. Thus, $d_{V_1'}(v) = d_{\bar{S}}(v) = d_{V_2'}(v)$ and so the vertices of S are co-satisfied. Given a vertex $v \in \bar{S}$, $d_{V_1'}(v) = 2d_S(v) + d_{\bar{S}}(v)$ while $d_{V_2'}(v) = d_{\bar{S}}(v)$, thus also the vertices of $\bar{S}$ are co-satisfied in G'.

Suppose now that G' is balanced co-satisfactory partitionable and let (V_1', V_2') be a balanced co-satisfactory partition. It is easy to see that F and T cannot be both included in the same part of the partition since otherwise the vertices of F and T are not co-satisfied. If the partition cuts only one of the two sets F or T, suppose for example that F is cut, then the vertices of F that are in the same part of the partition as T are not co-satisfied. If the partition cuts both F and T, denote by F_1, T_1 and F_2, T_2 the sets of vertices of F and T that are included in V_1' and V_2' respectively. For vertices of T_1 to be co-satisfied, we first have $|F_1| \leq |F_2|$ whereas for vertices of T_2 to be co-satisfied, we must have $|F_2| \leq |F_1|$, that is $|F_1| = |F_2|$, which is impossible since $|F|$ is odd. Therefore, F and T are included in different parts of the partition and thus (V_1', V_2') cuts the set V into two balanced sets V_1, V_2, where $V_1' = F \cup V_1$ and $V_2' = T \cup V_2$. We show that V_1 is an independent set. A vertex $v \in V_1$ has $d_{V_1'}(v) = 2d_{V_1}(v) + d_{V_2}(v)$ and $d_{V_2'}(v) = d_{V_2}(v)$. Since v is co-satisfied in G' we have $d_{V_1'}(v) \leq d_{V_2'}(v)$ and we obtain that $d_{V_1}(v) = 0$. Thus V_1 is an independent set of size $\frac{n}{2}$. □

Theorem 2. Satisfactory Partition *and* Balanced Satisfactory Partition *are NP-complete.*

Proof. Clearly, these two problems are in *NP*. We reduce Balanced Co-Satisfactory Partition to Balanced Satisfactory Partition which shows the *NP*-completeness of the latter problem by Theorem 1. Proposition 1 implies the *NP*-completeness of Satisfactory Partition. The reduction is as follows.

Let G be a graph, instance of Balanced Co-Satisfactory Partition on n vertices $v_1, \ldots, v_n$. The graph G', instance of Balanced Satisfactory Partition, has $2n$ vertices $v_1, \ldots, v_n, u_1, \ldots, u_n$. G' is the complement of graph G on vertices $v_1, \ldots, v_n$, and we add pendant edges (u_i, v_i), $i = 1, \ldots, n$. If G is balanced co-satisfactory partitionable and (V_1, V_2) is such a partition, then $V_i' = V_i \cup \{u_j : v_j \in V_i\}$ is a balanced satisfactory partition for G'. Indeed, if $v_i \in V_1$ then $d_{V_1}(v_i) \leq d_{V_2}(v_i)$ in G. Thus, in G' we have $d_{V_1'}(v_i) = \frac{n}{2} - 1 - d_{V_1}(v_i) + 1 \geq \frac{n}{2} - d_{V_2}(v_i) = d_{V_2'}(v_i)$ and $d_{V_1'}(u_i) = 1 > d_{V_2'}(u_i) = 0$. Conversely, since in each balanced satisfactory partition of G', u_i is in the same set as v_i, such a partition of G' gives a balanced co-satisfactory partition in G. □

4 No PTAS for Max (Co-)Satisfying Balanced Partition

In this section we prove that Max Co-Satisfying Balanced Partition and Max Satisfying Balanced Partition have no polynomial-time approximation scheme unless *P=NP*. We first introduce a problem used in our reductions.

Max k-Vertex Cover-B
Input: A graph $G = (V, E)$ with $|V| \geq k$ and maximum degree at most B.
Output: The maximum number of edges in G that can be covered by a subset $V' \subseteq V$ of cardinality k.

Theorem 3 (Petrank [7]). *There exists a constant α, $0 < \alpha < 1$ with the following property: given a graph G with n vertices and m edges, instance of* Max k-Vertex Cover-B *for some $k = \Theta(n)$, it is NP-hard to distinguish, whether it has $opt(G) = m$ or $opt(G) < (1 - \alpha)m$.*

Though it is not explicitly mentioned in [7], the proof of Theorem 3 yields the same conclusion for the restricted class of graphs with $m \geq \frac{n}{2}$. We prove next that the previous result holds in particular for $k = \frac{n}{2}$.

Theorem 4. *There exists a constant β, $0 < \beta < 1$, with the following property: given a graph G with N vertices and M edges, instance of* Max $\frac{N}{2}$-Vertex Cover-B′*, it is NP-hard to distinguish whether it has $opt(G) = M$ or $opt(G) < (1 - \beta)M$.*

Proof. We construct a gap-preserving reduction from Max k-Vertex Cover-B with $k = cn$, for some constant $c < 1$, to Max $\frac{N}{2}$-Vertex Cover-(2B+2). Let $G = (V, E)$ be a graph on n vertices and $m \geq \frac{n}{2}$ edges, instance of Max k-Vertex Cover-B. We will construct a graph G'' with N vertices and M edges such that if $opt(G) = m$ then $opt(G'') = M$ and if $opt(G) \leq (1 - \alpha)m$, for some $\alpha > 0$, then $opt(G'') \leq (1 - \beta)M$, for some $\beta > 0$.

First assume that $c > 1/2$. Let G'' be the graph obtained from G by inserting $2k - n$ isolated vertices. In this case, the properties of the gap-preserving reduction hold with $\beta = \alpha$.

Consider now the case $c < 1/2$. Suppose first that $n - 2k$ is a multiple of $B + 1$. Let G'' be the graph that consists of a copy of G and $\frac{n-2k}{B+1}$ copies of the graph T_{B+1} which is the complete tripartite graph whose vertex classes have cardinality $B + 1$ each. Observe that T_{B+1} needs $2B + 2$ vertices in covering its edges (the complement of a vertex class), and if just $2B + 2 - t$ vertices are taken, then at least $t(B + 1)$ edges remain uncovered. Thus, since G has maximum degree at most B, each subset of $\frac{N}{2}$ vertices not covering all copies of T_{B+1} is trivially improvable. Suppose first that $opt(G) = m$ and let V' be a vertex cover of size k in G. Then the set V' and the vertices of two among the 3 independent sets of each of the $\frac{n-2k}{B+1}$ copies of T_{B+1} form a vertex cover of G'' of size $\frac{N}{2}$, and thus $opt(G'') = M$. On the other hand, suppose $opt(G) < (1 - \alpha)m$. Then since $M = m + 3(B + 1)(n - 2k)$ and $m \geq \frac{n}{2}$, the number of edges not covered in G'' is at least $\alpha m \geq \frac{\alpha M}{1+6(B+1)(1-2c)}$ that can be viewed as βM.

Finally, if $c < 1/2$ and if $n-2k = \ell \ mod(B+1)$, $0 < \ell \leq B$, then let G' be the graph G together with further $B+1-\ell$ isolated vertices. Now, we can transform G' to G'' as before by inserting $\frac{n-2k-\ell}{B+1}+1$ copies of T_{B+1}. In this case we get a slightly different value for β, as the number m of edges is now compared with the modified number $n+B+1-\ell$ of vertices. Nevertheless, $\beta > 0$ is obtained. □

From this theorem, the following non-approximability results can be deduced.

Theorem 5. Max Co-Satisfying Balanced Partition *has no polynomial-time approximation scheme unless P=NP.*

Proof. We construct a gap-preserving reduction between Max $\frac{n}{2}$-Vertex Cover-B and Max Co-Satisfying Balanced Partition. Let G be a graph instance of Max $\frac{n}{2}$-Vertex Cover-B on n vertices and m edges. We construct the graph G' as in the proof of Theorem 1. Denote by N the number of vertices of G'.

Suppose first that $opt(G) = m$, and let V' be a vertex cover of size $\frac{n}{2}$ of G. Then in the partition $(F \cup (V \setminus V'), T \cup V')$ all vertices are co-satisfied and thus $opt(G') = N$.

Suppose now that $opt(G) < (1-\beta)m$. Thus for any set of $\frac{n}{2}$ vertices V', at least βm edges of G remain uncovered. The number of vertices incident to a non-covered edge is at least $\frac{2\beta m}{B}$. These vertices are not co-satisfied in the partition $(F \cup (V \setminus V'), T \cup V')$ and thus the number of co-satisfied vertices in this partition is less than $N - \frac{2\beta m}{B}$. It is lengthy but not too hard to show that, when a balanced partition cuts F or/and T, at least cm vertices are not co-satisfied, for some constant $c < 1$, and thus in this case we have $opt(G') < N - dm$, for some constant d. Since Max k-Vertex Cover-B is trivial for $m \leq k$, we may assume that $m \geq \frac{n}{2}$. Thus, since the number of vertices of G', $N = 4m + 2 + n \leq 7m$, we obtain $opt(G') < (1 - \frac{d}{7})N$. □

Theorem 6. Max Satisfying Balanced Partition *has no polynomial-time approximation scheme unless P=NP.*

Proof. Consider the graph G' with N vertices and M edges obtained in the construction given in the proof of Theorem 5, and apply to G' the reduction given in Theorem 2. Let G'' be the graph obtained. It can be shown that if $opt(G') = N$ then $opt(G'') = 2N$ and if $opt(G') < (1-\gamma)N$ then $opt(G'') < 2N(1 - c\gamma)$ for some constant c. □

5 Constant Approximations for Max (Co-)Satisfying Balanced Partition

We concentrate mostly on the approximation of Max Satisfying Balanced Partition. The co-satisfying version turns out to be simpler, and will be considered at the end of the section.

Proposition 2. *Any graph G with an odd number of vertices n has an almost balanced partition such that each vertex in the part of size $\frac{n+1}{2}$ is satisfied, and such a partition can be found in polynomial time.*

Proof. Let (V_1, V_2) be an almost balanced partition of G with $|V_1| > |V_2|$. If V_1 contains a vertex v that is not satisfied, then $d_{V_1}(v) < d_{V_2}(v)$ and thus by moving v from V_1 to V_2 we obtain an almost balanced partition with a smaller value of the cut induced by (V_1, V_2). The algorithm repeats this step while the largest set contains a non-satisfied vertex. After at most $|E|$ steps we obtain an almost balanced partition where the largest set contains only satisfied vertices. □

We consider now graphs of even order. Given a graph on an even number of vertices n, a vertex of degree $n-1$ is never satisfied in a balanced partition since it has only $\frac{n}{2} - 1$ neighbors in its own part and $\frac{n}{2}$ neighbors in the other part.

Theorem 7. *In any graph G on an even number of vertices n, a balanced partition with at least $\lceil \frac{n-t}{3} \rceil$ satisfied vertices can be found in polynomial time, where t is the number of vertices of degree $n-1$ in G.*

Proof. If G is not connected, then we find an almost balanced partition in each odd connected component, using Proposition 2, and a balanced partition in each even connected component (as shown afterwards); and then it is easy to put them together in order to form a balanced partition of G, where at least $\lceil \frac{n-t}{3} \rceil$ vertices are satisfied.

Suppose in the following that G is connected, and let H be the complement of G. Let $H_1, \ldots, H_q$ ($q \geq 1$) be the connected components of H. Observe that if a vertex is of degree $n-1$ in G then it forms alone a connected component in H. Denote by n_i the number of vertices of H_i, $i = 1, \ldots, q$. Consider now a connected component H_i, where $n_i > 1$. We will show that a (almost) balanced partition of $V(H_i)$ can be constructed where at least $\lceil \frac{n_i}{3} \rceil$ vertices are satisfied in G. (For n_i odd and n even, the almost balanced partition found in Proposition 2 may not work, since its smaller part will be completed with too many, $\frac{n}{2} - \frac{n_i - 1}{2}$ vertices in G.)

Let $M = \{(a_1, b_1), \ldots, (a_p, b_p)\}$ be a maximum matching in H_i. It can be found efficiently, using e.g. Edmonds' algorithm [2]. We distinguish two cases.

If $|M| \geq \lceil \frac{n_i}{3} \rceil$ then consider a (almost) balanced partition of the vertices of $V(H_i)$ except the vertices of the matching M. Let (V_1, V_2) be the partition of $V(H_i)$ obtained from this one by adding vertices a_j to V_1 and vertices b_j to V_2. While there exists a pair (a_j, b_j) where both vertices are not satisfied (in G), we exchange these two vertices. Since a_j and b_j are not linked in G, this exchange makes both a_j and b_j satisfied and decreases the value of the cut by at least 2. Therefore, after at most $\frac{|E|}{2}$ exchanges, we obtain a (almost) balanced partition with at least $\lceil \frac{n_i}{3} \rceil$ vertices satisfied (at least one vertex in each pair (a_j, b_j)).

If $|M| < \lceil \frac{n_i}{3} \rceil$ then using Gallai's decomposition theorem [3] we can obtain in polynomial time a vertex set S such that $2|M| = n_i - \ell + |S|$ where ℓ is the number of odd connected components of $H_i - S$. Let $O_1, \ldots, O_\ell$ be the odd connected components of $H_i - S$. Thus $\ell - |S| \geq \lceil \frac{n_i}{3} \rceil$ and so $\ell \geq \lceil \frac{n_i}{3} \rceil$ and $|S| \leq \lceil \frac{n_i}{3} \rceil$. Let us consider a vertex $v_j \in O_j$ linked to a vertex of S, for $j = 1, \ldots, \ell$. Those v_j are mutually adjacent in G.

If $\ell \geq \lceil \frac{n_i}{2} \rceil$ then we consider the following (almost) balanced partition (V_1, V_2): V_1 contains $\lceil \frac{n_i}{2} \rceil$ vertices from $v_1, \ldots, v_\ell$ and V_2 contains the other vertices. It is easy to see that at least $\lceil \frac{n_i}{2} \rceil$ vertices are satisfied in G, since for $v_j \in V_1$ we have $d_{V_1}(v_j) = \lceil \frac{n_i}{2} \rceil - 1$ and $d_{V_2}(v_j) \leq \lfloor \frac{n_i}{2} \rfloor - 1$.

Suppose next that $\lceil \frac{n_i}{3} \rceil \leq \ell \leq \lceil \frac{n_i}{2} \rceil$. If n_i is even, we construct a balanced partition (V_1', V_2') where V_1' contains $v_1, \ldots, v_\ell$ and V_2' contains S and $\ell - |S|$ other vertices; and if n_i is odd, we construct an almost balanced partition (V_1', V_2') where V_1' contains $v_1, \ldots, v_\ell$ and V_2' contains S and $\ell - 1 - |S|$ other vertices. In this latest step we pay attention, when we take some vertices from the remaining vertices of an odd connected component O_j, to take always an even number of vertices. In order to obtain a (almost) balanced partition (V_1, V_2) from (V_1', V_2') we consider the remaining vertices of each odd connected component O_j and we put half of these vertices in V_1 and half in V_2 such that v_j is satisfied. The partition in O_j does not influence the satisfied status of v_s for $s \neq j$, therefore it can be done independently in all O_j. We complete this partition by putting half of the remaining vertices in V_1 and half in V_2. □

Theorem 8. Max Satisfying Balanced Partition *is 3-approximable.*

Proof. Given a graph on n vertices, the maximum number of vertices that are satisfied in a balanced partition is $opt(G) \leq n - t$, where t is the number of vertices of degree $n-1$. Using Theorem 7 we obtain in polynomial time a balanced partition where the number of satisfied vertices is $val \geq \lceil \frac{n-t}{3} \rceil \geq \frac{opt(G)}{3}$. □

Theorem 9. Max Co-Satisfying Balanced Partition *is 2-approximable.*

Proof. Let (V_1, V_2) be a balanced partition of G. While there exists $v_1 \in V_1$ and $v_2 \in V_2$ that are not co-satisfied, we exchange v_1 and v_2. After this exchange the value of the cut increases by at least 2. Thus, after $\frac{|E|}{2}$ steps we obtain a balanced partition where at least one of the two parts contains co-satisfied vertices only. □

References

1. C. Bazgan, Zs. Tuza and D. Vanderpooten, *On the existence and determination of satisfactory partitions in a graph*, Proc. ISAAC 2003, LNCS 2906, 444–453.
2. J. Edmonds, *Paths, trees, and flowers*, Canadian J. of Math. 17 (1965), 449–467.
3. T. Gallai, *Maximale Systeme unabhängiger Kanten*, Magyar Tud. Akad. Mat. Kutató Int. Közl. 9 (1964), 401–413.
4. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, 1979.
5. M. Gerber and D. Kobler, *Partitioning a graph to satisfy all vertices*, Technical report, Swiss Federal Institute of Technology, Lausanne, 1998.
6. M. Gerber and D. Kobler, *Algorithmic approach to the satisfactory graph partitioning problem*, European Journal of Operation Research 125 (2000), 283–291.
7. E. Petrank, *The hardness of approximation: gap location*, Computational Complexity 4 (1994), 133–157.
8. K. H. Shafique and R. D. Dutton, *On satisfactory partitioning of graphs*, Congressus Numerantium 154 (2002), 183–194.

Distributed Weighted Vertex Cover via Maximal Matchings*

Fabrizio Grandoni[1], Jochen Könemann[2], and Alessandro Panconesi[1]

[1] Dipartimento di Informatica, Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
{grandoni,ale}@di.uniroma1.it
[2] Department of Combinatorics and Optimization, University of Waterloo
200 University Avenue West, Waterloo, ONN2L 3G1, Canada
jochen@math.uwaterloo.ca

Abstract. In this paper we consider the problem of computing a minimum-weight vertex-cover in an n-node, weighted, undirected graph $G = (V,E)$. We present a fully distributed algorithm for computing vertex covers of weight at most twice the optimum, in the case of integer weights. Our algorithm runs in an expected number of $O(\log n + \log \hat{W})$ communication rounds, where $\hat{W}$ is the average vertex-weight. The previous best algorithm for this problem requires $O(\log n(\log n + \log \hat{W}))$ rounds and it is not fully distributed.
For a maximal matching M in G it is a well-known fact that any vertex-cover in G needs to have at least $|M|$ vertices. Our algorithm is based on a generalization of this combinatorial lower-bound to the weighted setting.

1 Introduction

We are given an undirected graph $G = (V,E)$ and non-negative vertex weights $w_v \leq W$ for all vertices $v \in V$. A *vertex cover* is a subset $C \subseteq V$ such that each edge $e \in E$ has at least one end-point in C. In the *minimum-weight vertex-cover* problem we want to compute a vertex-cover of smallest total weight.

Computing minimum-weight vertex-covers is NP-hard [4]. Papadimitriou and Yannakakis [14] show that the problem is APX-hard. More recently, Håstad [8] proves that there is no $(7/6-\varepsilon)$-approximation algorithm for the vertex-cover problem for any $\varepsilon > 0$ unless P = NP.

On the positive side, the best known algorithms for the vertex cover problem are due to Bar-Yehuda and Even [1], and to Monien and Speckenmeyer [12]. These algorithms achieve an approximation ratio of $(2 - \frac{\log\log n}{2\log n})$. In graphs with maximum degree Δ, Hochbaum [9] gives a $(2-1/\Delta)$-approximation algorithm for the problem. This was subsequently improved by Halldórsson and Radhakrishnan [5] who present a $(2 - \frac{\log(\Delta)+O(1)}{\Delta})$-approximation algorithm. Finally, Halperin [6] presents the currently best algorithm for the problem with a performance ratio of $(2-(1-o(1))\frac{2\ln\ln\Delta}{\ln\Delta})$.

In the distributed setting, it is known how to compute a 2-approximate vertex cover in the unweighted case. This can be achieved by computing a maximal matching in

* The first and third authors were supported with funding from EC Projects DELIS and EYES, and project WebMinds of the Italian Ministry of University and Research (MIUR)

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 839–848, 2005.

the graph and by including the matched nodes in the cover. A maximal matching can be computed in $O(\log^4 n)$ rounds via the algorithm of Hanckowiack et al. [7], and in $O(\Delta + \log^* n)$ rounds via the algorithm of Panconesi and Rizzi [13]. Both algorithms are deterministic. Maximal matchings can also be computed in an expected number of $O(\log n)$ rounds via the randomized algorithm of Israeli and Itai [10].

For the weighted case a $(2+\varepsilon)$-approximation can be computed deterministically in $O(\log n \log \frac{1}{\varepsilon})$ many rounds by using the algorithm of Khuller et al. [11]. Their algorithm is stated as a PRAM algorithm, but it is readily seen to be a bona fide distributed algorithm. Let $\hat{W}$ be the average weight. Then, by setting $\varepsilon = 1/(n\hat{W}+1)$, the latter algorithm computes a 2-approximate vertex cover in $O(\log n(\log n + \log \hat{W}))$ communication rounds. Note that the above choice of ε requires global knowledge of the quantity $n\hat{W}$. This assumption may not be realistic in all scenarios.

In this paper we present an improved fully-distributed algorithm to compute a 2-approximate weighted vertex cover, in the case of integer weights. Our main result can be stated as follows:

Theorem 1. *There is a fully distributed algorithm which computes a 2-approximate weighted vertex cover in an expected number of* $O(\log n + \log \hat{W})$ *communication rounds. The message size is* $O(\log W)$ *and the local computation done in each round is* $O(\Delta \log(\Delta W))$ *in expectation.*

Our algorithm can be viewed as a generalization of the reduction from unweighted vertex cover to maximal matching. The basic idea is to expand each node v of weight w_v into w_v *micro-nodes* $v(1), v(2) \ldots, v(w_v)$, and connect each $v(i)$ to every $u(j)$ whenever vu is an edge of the network. Then a maximal matching in the auxiliary graph is computed. The vertex cover is given by the nodes such that all the corresponding micro-nodes are matched. If the maximal matching is computed via the fully-distributed algorithm of Israeli and Itai, the algorithm halts in an expected number of $O(\log n + \log \hat{W})$ rounds.

A naive implementation of the matching algorithm by Israeli and Itai leads to pseudo-polynomial message and time complexity in each round. The main insight leading to the bounds on message-size and local computation time in Theorem 1 is to keep an implicit representation of the auxiliary graph and a maximal matching in it.

The rest of this paper is organized as follows. In Section 2 we introduce some preliminaries. Our algorithm relies on a careful adaptation of the matching algorithm by Israeli and Itai. We present this adaptation in Section 3. Finally, Sections 4 and 5 deal with the naive and refined versions of our weighted vertex cover algorithm, respectively.

2 Preliminaries

The minimum-weight vertex cover problem can be formulated as an *integer linear program* (ILP):

$$\begin{array}{lll} \min & \sum_{v \in V} w_v x_v & \\ \text{s.t.} & & \\ x_v + x_u \geq 1, & & \forall vu \in E; \\ x_v \in \{0,1\}, & & \forall v \in V. \end{array}$$

Each assignment of the variables which satisfies the constraints (*feasible solution*) corresponds to a vertex cover containing exactly the nodes v with $x_v = 1$. By (LP) we denote the natural *linear programming relaxation* of (ILP).

Let $N(v)$ be the set of neighbors of v. The *linear programming dual* (D) of (LP) is:

$$\begin{array}{ll} \max & \sum_{vu \in E} y_{vu} \\ \text{s.t.} & \\ \sum_{u \in N(v)} y_{vu} \leq w_v, & \forall v \in V; \\ y_{vu} \geq 0, & \forall vu \in E. \end{array}$$

By *weak duality* (e.g., see [2]), the value of each feasible solution of (D) is a lower bound for the value of every feasible solution of (LP) and hence (IPL).

In this paper we consider a synchronous message-passing model of computation. The computation proceeds in rounds. In each round, a node can send/receive a message (of unbounded size) to/from each one of its neighbors, and execute an unbounded amount of computation. No global knowledge is available (including the number n of nodes in the graph). The algorithms presented can be easily modified so as to work in a (non-faulty) asynchronous system also.

By $B(p)$, $p \in [0,1]$, we denote a 0-1 *Bernoulli* random variable, which takes value one with probability p. A *random bit* is a Bernoulli variable $B(0.5)$.

3 Distributed Maximal Matching

A *matching* of a graph $G = (V,E)$ is a subset $M \subseteq E$ such that no two edges of M are incident to the same node. The results of the next sections are based on the following simplified version $\mathcal{M}$ of the distributed maximal-matching algorithm of Israeli and Itai [10].

Algorithm $\mathcal{M}$ works in phases, each one consisting of a constant number of rounds. In each phase, a matching is computed and the edges incident on matched nodes are removed. The algorithm halts when no edge is left. The maximal matching is given by the union of the matchings found in the different phases.

In a given phase a matching is computed in the following way. Let $G' = (V',E')$ be the current graph. By $N'(v)$ and δ'_v we denote the set of neighbors of v and the degree of v in G', respectively. Each node v randomly decides to be a *sender* or a *receiver* with probability one half. Note that the same node may play a different role in different phases. Each sender u selects one neighbor $v \in N'(u)$ uniformly at random and makes a *proposal* to v. Each receiver v which receives at least one proposal, selects one of the proponents (arbitrarily) and *accepts* its proposal. The matching is given by the edges corresponding to accepted proposals.

Let a node v be *good* if at least one third of its neighbors u have degree $\delta_u \leq \delta_v$. To prove the bound on the number of rounds, we use the following simple combinatorial result [10]:

Lemma 1. *At least one half of the edges of a graph are incident to good nodes.*

Theorem 2. *Algorithm $\mathcal{M}$ computes a maximal matching in $O(\log n)$ expected rounds.*

Proof. The correctness of the algorithm is trivial.

We show that in each phase at least a constant fraction of the edges is removed in expectation. This implies that the expected number of rounds is $O(\log(n^2)) = O(\log n)$. Consider a good node v of G' in a given phase. The probability P'_v that v accepts a proposal is lower bounded by:

$$P'_v \geq \frac{1}{2}\left(1 - \prod_{u \in N'(v)} \left(1 - \frac{1}{2\delta'_u}\right)\right).$$

From the definition of good nodes:

$$\prod_{u \in N'(v)} \left(1 - \frac{1}{2\delta'_u}\right) \leq \prod_{u \in N'(v):\delta'_u \leq \delta'_v} \left(1 - \frac{1}{2\delta'_v}\right) \leq \left(1 - \frac{1}{2\delta'_v}\right)^{\delta'_v/3} \leq e^{-1/6}.$$

Thus $P'_v \geq (1 - e^{-1/6})/2$ and the expected number of edges removed is at least a fraction $(1 - e^{-1/6})/4$ of the total. □

4 Distributed Vertex Cover via Maximal Matchings

In this section we present a simple pseudo-polynomial reduction from the problem of computing a 2-approximate vertex cover to the problem of computing a maximal matching in an auxiliary graph. Thanks to this reduction, a 2-approximate vertex cover can be computed in $O(\log n + \log \hat{W})$ expected rounds via algorithm $\mathcal{M}$ of section 3.

Consider the following auxiliary graph $\widetilde{G}$. For each node v of G, $\widetilde{G}$ contains w_v *micro-nodes* $v(1), v(2) \dots v(w_v)$. Two micro-nodes $v(i)$ and $u(j)$ are adjacent if and only if vu is an edge of G. In Figure 1 an example of the reduction is given.

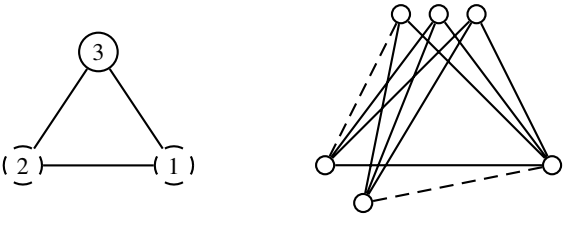


Fig. 1. A weighted graph G (on the left) with the corresponding auxiliary graph $\widetilde{G}$. A maximal matching M of $\widetilde{G}$ is indicated via dashed lines. The dashed nodes of G form a vertex cover

Let M be a maximal matching in $\widetilde{G}$. By $V(M)$ we denote the set of nodes v of G such that all the corresponding micro-nodes $v(i)$ are matched by M.

Lemma 2. *Set $V(M)$ is a 2-approximate vertex cover of G.*

Proof. Assume by contradiction that $V(M)$ is not a vertex cover. Thus there are two adjacent nodes v and u in G which do not belong to $V(M)$. This implies that there are two adjacent micro-nodes $v(i)$ and $u(j)$ in $\widetilde{G}$ which are not matched by M. Then the set $M' = M \cup \{v(i)u(j)\}$ is a matching, which contradicts the maximality of M.

Let apx and opt denote the weight of the vertex cover found and that of a minimum weight vertex cover, respectively. Moreover, let z_v be the number of micro-nodes in $\{v(1), v(2) \dots v(w_v)\}$ that are matched by M. A feasible solution of (D) is obtained by assigning to each dual variable y_{vu} the number of edges of the kind $v(i)u(j) \in M$. This solution is feasible since, for every $v \in V$: $\sum_{u \in N(v)} y_{vu} = z_v \leq w_v$. By weak duality we obtain $apx \leq \sum_{v \in V} z_v \leq 2 \sum_{vu \in E} y_{vu} \leq 2\, opt$ and hence $V(M)$ is 2-approximate. □

Lemma 2 suggests a strategy to compute a 2-approximate vertex cover distributively. The idea is to simulate the behavior of algorithm $\mathcal{M}$ on a virtual auxiliary graph $\widetilde{G}$, and then to select the nodes in the vertex cover as suggested by Lemma 2.

Specifically, each node simulates the execution of the algorithm on the corresponding micro-nodes $v(i)$ in $\widetilde{G}$. Whenever two micro-nodes $v(i)$ and $u(j)$ of $\widetilde{G}$ need to communicate, nodes v and u are responsible for allowing such communication. The vertex cover is given by the nodes v such that all the corresponding micro-nodes $v(i)$ are matched by the maximal matching computed. Since the virtual auxiliary graph contains $O(n\hat{W})$ nodes, the total number of rounds is $O(\log(n\hat{W})) = O(\log n + \log \hat{W})$.

This naive application of Lemma 2 has two major drawbacks. The first problem is the large message size. In fact, in each phase all the micro-nodes of v may send a proposal to some micro-node of u. Thus the message size is $\Omega(W)$.

A second problem is the time complexity of the algorithm: consider a node v in a given phase. Each micro-node $v(i)$ of v, with probability one half, needs to select one neighbor out of $\Theta(\Delta W)$ uniformly at random. This random selection can be performed in $\Theta(\log(\Delta W))$ expected time, assuming that the cost of generating a random bit is $O(1)$ (e.g., see [3]). Thus the expected time complexity of each phase is $\Omega(W \log(\Delta W))$.

In next section we show how to solve both problems by creating the matchings *implicitly*.

5 An Improved Algorithm

In this section we present an improved fully distributed algorithm $\mathcal{A}$ for computing a 2-approximate vertex cover. Algorithm $\mathcal{A}$ still requires $O(\log n + \log \hat{W})$ expected rounds, but it reduces the size of the messages to $O(\log W)$ and the expected time complexity of each phase to $O(\Delta \log(\Delta W))$.

The basic structure of algorithm $\mathcal{A}$ is analogous to the structure of the naive algorithm described in previous section: in each phase, a matching in the current auxiliary graph $\widetilde{G}'$ is computed, and the matched nodes are removed from $\widetilde{G}'$ (together with all the edges incident to them). The algorithm halts when no edge is left. The vertex cover is given by the nodes v such that all the corresponding micro-nodes $v(i)$ are matched by one of the matchings computed. The main novelty in Algorithm $\mathcal{A}$ is that matchings are created *implicitly*: in each phase each node only knows the number of the corresponding matched micro-nodes. Intuitively, this simplification is allowed by the symmetry properties of $\widetilde{G}$: all the micro-nodes corresponding to a node v have the same degree and share the same neighborhood. This invariant is kept by all the induced subgraphs of $\widetilde{G}$.

Algorithm $\mathcal{A}$, which is described in Figure 2, works in phases. Each phase consists of a constant number of communication rounds. Each node v has an associated *state*

$w'_v = w_v$; $N'(v) = N(v)$, s_v = *active*;
while (s_v = *active*) {
  send w'_v and **receive** w'_u **to/ from** all $u \in N'(v)$;
  $N'(v) = \{u \in N(v) : w'_u > 0\}$;
  if ($|N'(v)| = 0$)
    s_v = *outside*;
  else {
    compute the proposals $p_v(u)$;
    send $p_v(u)$ and **receive** $p_u(v)$ **to/ from** all $u \in N'(v)$;
    compute the counter-proposals $c_v(u)$;
    send $c_v(u)$ and **receive** $c_u(v)$ **to/ from** all $u \in N'(v)$;
    for (all $u \in N'(v)$)
      $w'_v = w'_v - c_v(u) - c_u(v)$;
    if ($w'_v = 0$) {
      send w'_v **to** all $u \in N'(v)$;
      s_v = *inside*;
    }
  }
}

Fig. 2. Protocol for node v for 2-approximate vertex cover

s_v, which is initially *active*. In each phase, part of the *active* nodes switch to the state *inside* or *outside*, and the algorithm terminates when no *active* node is left. When a node leaves the *active* state, it halts. At the end of the algorithm, the *inside* nodes form a vertex cover.

In more details, each node v has an associated *residual weight* w'_v, which is initially w_v. The residual weight w'_v can be interpreted as the number of micro-nodes $v(i)$ of v in the current auxiliary graph $\widetilde{G}'$. Note that all the micro-nodes $v(i)$ have the same degree $W'_v = \sum_{u \in N(v)} w'_u$. In each phase, the expected residual weight of active nodes decreases. The decrease of w'_v in a given phase reflects the number of micro-nodes of v that have been matched in that phase.

At the beginning of each phase, each *active* node v sends w'_v to all its currently active neighbors $N'(v)$. The neighbors with $w'_u = 0$ are removed from $N'(v)$. If $N'(v)$ becomes empty, node v switches to the *outside* state. In fact, in this case the degree W'_v of the micro-nodes $v(i)$ is zero, and thus they will never be matched.

Otherwise, v sends a *proposal* $p_v(u)$ to each *active* neighbor $u \in N'(v)$. The value of $p_v(u)$ can be interpreted as the number of proposals directed from the micro-nodes of v to the micro-nodes of u. Let p'_v be the sum of the proposals $p_v(u)$:

$$p'_v = \sum_{u \in N'(v)} p_v(u).$$

This quantity can be viewed as the number of *micro-senders* among $v(1), v(2) \ldots v(w_v)$. We postpone a detailed description of how proposals are fixed until later.

For each proposal $p_u(v)$ received, node v replies with a *counter-proposal* $c_v(u)$. The counter-proposal $c_v(u)$ can be interpreted as the number of micro-nodes of v which accept proposals of micro-nodes of u. Let $c'_v = w'_v - p'_v$ be the number of *micro-receivers*

of v. The sum of the counter-proposals for node v then needs to be at most c'_v. At the same time each counter-proposal $c_v(u)$ must not exceed the corresponding proposal $p_u(v)$. Given these restrictions we choose a feasible set of counter-proposals $\{c_v(u)\}_{u\in N'(v)}$ arbitrarily such that their sum is maximum, i.e.

$$\sum_{u\in N'(v)} c_v(u) = \min\left\{c'_v, \sum_{u\in N'(v)} p_u(v)\right\}.$$

Eventually, node v decrements w'_v by the sum of all the counter-proposals $c_v(u)$ and $c_u(v)$ which have been sent and received by v, respectively: $w'_v = w'_v - \sum_{u\in N'(v)}(c_v(u)+c_u(v))$. This decrement reflects the number of micro-nodes of v which are matched in the considered phase.

If w'_v becomes zero, node v sends w'_v to all its neighbors (for the last time) and switches to the *inside* state (since all the corresponding micro-nodes are matched).

We now show how proposals are fixed by each node v. If the number w'_v of micro-nodes $v(i)$ is "sufficiently" small, the proposals $p_v(u)$ are fixed according to algorithm $\mathcal{M}$. Otherwise, they are fixed in a more efficient way, while keeping the same expected value. In more details, there are two different strategies, depending on whether $w'_v < 2\delta'_v$ or not, where $\delta'_v = |N'(v)|$ is the number of currently *active* neighbors of v. If $w'_v < 2\delta'_v$, the proposals $p_v(u)$ are initially set to zero. Then, for w'_v times, an active neighbor $u \in N'(v)$ is selected at random with probability proportional to w'_u, and the corresponding proposal $p_v(u)$ is incremented by one with probability one half. Note that each $p_v(u)$, considered separately, is the sum of w'_v i.i.d. Bernoulli variables $B(w'_u/(2W'_v))$:

$$p_v(u) = \sum_{i=1}^{w'_v} B\left(\frac{w'_u}{2W'_v}\right). \tag{1}$$

Otherwise ($w'_v \geq 2\delta'_v$), the value of each $p_v(u)$ is independently set to:

$$p_v(u) = \left\lfloor \frac{w'_v w'_u}{2W'_v} \right\rfloor + B\left(\frac{w'_v w'_u}{2W'_v} - \left\lfloor \frac{w'_v w'_u}{2W'_v} \right\rfloor\right). \tag{2}$$

Note that, in both cases, the sum p'_v of the proposals is upper bounded by w'_v. In the first case this is trivially true. In the second case, this is a consequence of the small value of δ'_v:

$$\sum_{u\in N'(v)} p_v(u) \leq \sum_{u\in N'(v)} \left(\frac{w'_v w'_u}{2W'_v} + 1\right) = \frac{w'_v}{2} + \delta'_v \leq w'_v.$$

Moreover, in both cases $E[p'_v] = w'_v/2$. The following technical property of proposals will be useful in later parts of the analysis.

Lemma 3. *For any two given nodes v and $u \in N'(v)$ we have*

$$E_{u,v} = E\left[\left(1-\frac{1}{w'_v}\right)^{p_u(v)}\right] \leq \left(1-\frac{1}{W'_u}\right)^{w'_u/4}.$$

Proof. If $w'_u < 2\delta'_u$, by Equation (1):

$$E_{u,v} = E\left[\left(1-\frac{1}{w'_v}\right)^{\sum_{i=1}^{w'_u} B\left(\frac{w'_v}{2W'_u}\right)}\right] = \left(E\left[\left(1-\frac{1}{w'_v}\right)^{B\left(\frac{w'_v}{2W'_u}\right)}\right]\right)^{w'_u} =$$

$$= \left(1-\frac{w'_v}{2W'_u}+\frac{w'_v}{2W'_u}\left(1-\frac{1}{w'_v}\right)\right)^{w'_u} \leq \left(1-\frac{1}{W'_u}\right)^{w'_u/2}.$$

Consider now the case $w'_u \geq 2\delta'_u$. By Equation (2), if $w'_u w'_v/(2W'_u) \geq 1$:

$$E_{u,v} \leq E\left[\left(1-\frac{1}{w'_v}\right)^{\lfloor w'_u w'_v/(2W'_u)\rfloor}\right] \leq \left(1-\frac{1}{w'_v}\right)^{w'_u w'_v/(4W'_u)} \leq \left(1-\frac{1}{W'_u}\right)^{w'_u/4}.$$

Otherwise ($w'_u w'_v/(2W'_u) < 1$):

$$E_{u,v} = E\left[\left(1-\frac{1}{w'_v}\right)^{B\left(\frac{w'_u w'_v}{2W'_u}\right)}\right] = 1-\frac{w'_u}{2W'_u} \leq \left(1-\frac{1}{W'_u}\right)^{w'_u/2}.$$

□

Lemma 4. *Algorithm $\mathcal{A}$ computes a* 2*-approximate vertex cover.*

Proof. The algorithm halts. In fact, the residual weight of each *active* node decreases by at least one in each round with positive probability. It follows that the nodes which do not switch to the *outside* state, switch to the *inside* state in a finite expected number of rounds. Assume by contradiction that, at the end of the algorithm, the *inside* nodes do not form a vertex cover. This implies that there is an *outside* node v which has at least one *outside* neighbor. Let v switch to the state *outside* in phase p. At the beginning of phase $(p-1)$, all the neighbors of v are either *inside* or *active* nodes. Consider the *active* neighbors of v in phase $(p-1)$. These nodes are not active any more when phase p starts. But they cannot switch to the state *outside* in phase $(p-1)$, since their active degree is greater than zero in that phase. Thus they all switch to the state *inside*, which is a contradiction. Let z_v be the difference between w_v and the final residual weight w'_v. A feasible solution of (D) is obtained by assigning to each dual variable y_{vu} the sum of all the counter-proposals of the kind $c_v(u)$ and $c_u(v)$. Let `apx` and `opt` be the weight of the vertex cover found and that of a minimum vertex cover, respectively. By weak duality: $apx \leq \sum_{v\in V} z_v \leq 2\sum_{vu\in E} y_{vu} \leq 2\,opt$. Thus the vertex cover found is 2-approximate. □

Lemma 5. *Algorithm $\mathcal{A}$ sends messages of size* $O(\log W)$*. Each phase of algorithm $\mathcal{A}$ has time complexity* $O(\Delta \log(\Delta W))$ *in expectation.*

Proof. Both proposals and counter-proposals can be packed in messages of size $O(\log W)$. The time complexity of each phase is upper bounded by the cost of computing the proposals. Computing the proposals is as expensive as selecting $O(\Delta)$ times an element out of $O(\Delta W)$ ones uniformly at random. Each random selection can be

performed by generating $O(\log(\Delta W))$ random bits in expectation. By assuming a $O(1)$ cost for generating a random bit, the total expected cost of each phase is $O(\Delta \log(\Delta W))$. □

Recall that a node is *good* if at least one third of its neighbors have degree smaller or equal than its own degree. Consider a node v in G. The degree of all the micro-nodes corresponding to v in $\widetilde{G}$ is $W_v = \sum_{u \in N(v)} w_u$. Thus a micro-node $v(i)$ is good if and only if:

$$\sum_{u \in N(v): W_u \leq W_v} w_u \geq \frac{W_v}{3}.$$

Note that, if a micro-node $v(i)$ is good, all the micro-nodes $v(j)$, $j \in \{1, 2 \ldots w_v\}$, are good and vice-versa. We call a node of G *heavy* if all its micro-nodes in $\widetilde{G}$ are good. The next observation can be seen as the weighted analogue of Lemma 1.

Lemma 6. *Let $E_H \subseteq E$ be the subset of edges incident to heavy nodes. Then $\sum_{vu \in E_H} w_v w_u \geq \frac{1}{2} \sum_{vu \in E} w_v w_u$.*

Proof. Consider the auxiliary graph $\widetilde{G}$. The number of edges of $\widetilde{G}$ that are incident to good nodes is $\sum_{\{v,u\} \in E_H} w_v w_u$. Since the number of edges of $\widetilde{G}$ is $\sum_{\{v,u\} \in E} w_v w_u$, the claim follows from Lemma 1. □

We use the properties of heavy nodes to prove the following bound on the number of rounds.

Lemma 7. *Algorithm $\mathcal{A}$ halts in $O(\log n + \log \hat{W})$ expected rounds.*

Proof. We show that the residual weight of heavy nodes decreases by at least a positive constant factor in expectation in each phase. It follows from Lemma 6 that the same holds for the potential function: $0 \leq \sum_{vu \in E} w'_v w'_u < (n\hat{W})^2$, thus implying the claim.

Consider a heavy node v in a given phase. Let w''_v be the values of w'_v at the end of the phase. The residual weight of v decreases by at least the sum of the counter-proposals $c_v(u)$ sent by v:

$$\begin{aligned} w''_v \leq w'_v - \sum_{u \in N'(v)} c_v(u) &= p'_v + c'_v \left(1 - \frac{1}{c'_v} \min\{c'_v, \sum_{u \in N'(v)} p_u(v)\}\right) \\ &= p'_v + (w'_v - p'_v) \left(1 - \frac{1}{c'_v} \min\{c'_v, \sum_{u \in N'(v)} p_u(v)\}\right). \end{aligned}$$

Note that:

$$\left(1 - \frac{1}{c'_v} \min\{c'_v, \sum_{u \in N'(v)} p_u(v)\}\right) \leq \left(1 - \frac{1}{c'_v}\right)^{\sum_{u \in N'(v)} p_u(v)} \leq \prod_{u \in N'(v)} \left(1 - \frac{1}{w'_v}\right)^{p_u(v)}.$$

Thus:

$$E[w''_v] \leq \frac{w'_v}{2} + \frac{w'_v}{2} \prod_{u \in N'(v)} E\left[\left(1 - \frac{1}{w'_v}\right)^{p_u(v)}\right],$$

where we used $E[p'_v] = w'_v/2$. By Lemma 3 and the definition of heavy nodes:

$$\prod_{u\in N'(v)} E\left[\left(1-\frac{1}{w'_v}\right)^{p_u(v)}\right] \le \prod_{u\in N'(v)}\left(1-\frac{1}{W'_u}\right)^{\frac{w'_u}{4}}$$
$$\le \prod_{\substack{u\in N'(v)\\ W'_u\le W'_v}}\left(1-\frac{1}{W'_v}\right)^{\frac{w'_u}{4}} \le \left(1-\frac{1}{W'_v}\right)^{\frac{W'_v}{12}}.$$

The right-hand side is at most $e^{-1/12}$ and it follows that $E[w''_v] \le w'_v(1+e^{-1/12})/2$. □

Lemmas 4, 5, and 7 together imply Theorem 1.

References

1. R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.
2. V. Chvátal. *Linear programming*. Freeman, 1983.
3. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill Book Company, 6th edition, 1992.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. Freemann, 1979.
5. M. M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. In *ACM Symposium on the Theory of Computing*, pages 439–448, 23–25 1994.
6. E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623, Oct. 2002.
7. M. Hańćkowiak, M. Karoński, and A. Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics*, 15(1):41–57, 2001.
8. J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, July 2001.
9. D. S. Hochbaum. Efficient bounds for the stable set, vertex cover, and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.
10. A. Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22:77–80, 1986.
11. S. Khuller, U. Vishkin, and N. Young. A primal-dual parallel approximation technique applied to weighted set and vertex cover. *Journal of Algorithms*, 17(2):280–289, 1994.
12. B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22:115–123, 1985.
13. A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *DISTCOMP: Distributed Computing*, 14, 2001.
14. C. Papadimitriou and M. Yannakakis. Optimization, approximization and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

On the Complexity of the Balanced Vertex Ordering Problem*

Jan Kára[1], Jan Kratochvíl[1], and David R. Wood[2]

[1] Department of Applied Mathematics, Faculty of Mathematics and Physics
Charles University, Prague, Czech Republic
{kara,honza}@kam.mff.cuni.cz
[2] Departament de Matemàtica Aplicada II
Universitat Politècnica de Catalunya, Barcelona, Spain
david.wood@upc.edu

Abstract. We consider the problem of finding a balanced ordering of the vertices of a graph. More precisely, we want to minimise the sum, taken over all vertices v, of the difference between the number of neighbours to the left and right of v. This problem, which has applications in graph drawing, was recently introduced by Biedl *et al.* [1]. They proved that the problem is solvable in polynomial time for graphs with maximum degree three, but $\mathcal{NP}$-hard for graphs with maximum degree six. One of our main results is closing the gap in these results, by proving $\mathcal{NP}$-hardness for graphs with maximum degree four. Furthermore, we prove that the problem remains $\mathcal{NP}$-hard for planar graphs with maximum degree six and for 5-regular graphs. On the other hand we present a polynomial time algorithm that determines whether there is a vertex ordering with total imbalance smaller than a fixed constant, and a polynomial time algorithm that determines whether a given multigraph with even degrees has an 'almost balanced' ordering.

1 Introduction

A number of algorithms for graph drawing use a 'balanced' ordering of the vertices of the graph as a starting point [2–4, 6, 7]. Here balanced means that neighbours of each vertex v are as evenly distributed to the left and right of v as possible (see below for more precise definition). The problem of determining such an ordering was recently studied by Biedl *et al.* [1]. We solve a number of open problems from [1] and study a few other related problems.

Let $G = (V, E)$ be a multigraph without loops. An *ordering* of G is a bijection $\sigma : V \to \{1, \ldots, |V|\}$. For $u, v \in V$ with $\sigma(u) < \sigma(v)$, we say that u is *to the left*

* Supported by grant MEC SB2003-0270. Research completed at the Department of Applied Mathematics and the Institute for Theoretical Computer Science, Charles University, Prague, Czech Republic. Supported by projects LN00A056 and 1M0021620808 of the Ministry of Education of the Czech Republic, and by the European Union Research Training Network COMBSTRU (Combinatorial Structure of Intractable Problems)

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 849–858, 2005.

of v and that v is *to the right* of u. The *imbalance* of $v \in V$ in σ, denoted by $B_\sigma(v)$, is

$$\big||\{e \in E : e = \{u, v\}, \sigma(u) < \sigma(v)\}| - |\{e \in E : e = \{u, v\}, \sigma(u) > \sigma(v)\}|\big|.$$

When the ordering σ is clear from the context we simply write $B(v)$ instead of $B_\sigma(v)$. The *imbalance* of ordering σ, denoted by $B_\sigma(G)$, is $\sum_{v \in V} B_\sigma(v)$. The minimum value of $B_\sigma(G)$, taken over all orderings σ of G, is denoted by $M(G)$. An ordering with imbalance $M(G)$ is called *minimum*. Clearly the following two facts hold for any ordering:

- Every vertex of odd degree has imbalance at least one.
- The two vertices at the beginning and at the end of any ordering have imbalance equal to their degrees.

These two facts imply the following lower bound on the imbalance of an ordering. Let $\mathrm{odd}(A)$ denote the number of odd degree vertices among the vertices of $A \subseteq V$. Let $(d_1, \ldots, d_n)$ be the sequence of vertex degrees of G, where $d_i \leq d_{i+1}$ for all $1 \leq i \leq n-1$. Then

$$B_\sigma(G) \geq \mathrm{odd}(V) - (d_1 \bmod 2) - (d_2 \bmod 2) + d_1 + d_2.$$

An ordering σ is *perfect* if the above inequality holds with equality. PERFECT ORDERING is the decision problem whether a given multigraph G has a perfect ordering. This problem is clearly in $\mathcal{NP}$.

Biedl *et al.* [1] gave a polynomial time algorithm to compute a minimum ordering of graphs with maximum degree at most three. On the other hand, they proved that it is $\mathcal{NP}$-hard to compute a minimum ordering of a (bipartite) graph with maximum degree six.

One of the main results of this paper is to close the above gap in the complexity of the balanced ordering problem with respect to the maximum degree of the graph. In particular, we prove that the PERFECT ORDERING problem is $\mathcal{NP}$-complete for simple graphs with maximum degree four.

Whether the balanced ordering problem is efficiently solvable for planar graphs is of particular interest since planar graphs are often used in graph drawing applications. We answer this question in the negative by proving that the PERFECT ORDERING problem is $\mathcal{NP}$-complete for planar simple graphs with maximum degree six.

Our third $\mathcal{NP}$-hardness result states that finding a minimum ordering is $\mathcal{NP}$-hard for 5-regular simple graphs. All of these $\mathcal{NP}$-hardness results are presented in Section 3. The proofs are based on reductions from various satisfiability problems. Section 2 contains several $\mathcal{NP}$-completeness results for the satisfiability problems that we use.

In Section 4 we present our positive complexity results. In particular, we describe a polynomial time algorithm that determines whether a given graph has an ordering with at most k imbalanced vertices for any constant k. This algorithm has several interesting corollaries. For example, the PERFECT ORDERING problem can be solved in polynomial time for a multigraph in which all the vertices have even degrees (in particular, for 4-regular multigraphs).

2 $\mathcal{NP}$-Hardness of Satisfiability Problems

In this section we state several $\mathcal{NP}$-hardness results about various satisfiability problems. The results in this section can be achieved by verifying conditions of a general theorem of Schaefer [5]. First we introduce several basic definitions about satisfiability. Throughout this paper, formulae are considered to be in a *conjunctive normal form*. We allow a variable to occur several times in one clause but note that the graphs created in this way are simple (unless stated otherwise). Suppose φ is a formula with variables $x_1, \ldots, x_n$. The *incidence graph* of φ is the bipartite graph with vertices $c_1, \ldots, c_m$ and $x_1, \ldots, x_n$, where $\{c_i, x_j\}$ is an edge if and only if the variable x_j occurs in the clause c_i (positive or negated). A *truth assignment* of a formula φ with variables $x_1, \ldots, x_n$ is an arbitrary function $t : \{1, \ldots, n\} \to \{0, 1\}$. The values 0 and 1 are also sometimes called *false* and *true* respectively. A truth assignment t is *satisfying* φ if there is at least one true literal in every clause. The formula φ is *satisfiable* if it has at least one satisfying truth assignment.

The decision problem asking whether a given formula φ is satisfiable is called SAT. If we assume that every clause in the given formula φ has size exactly three, then the decision problem asking whether φ is satisfiable is called 3SAT. Two common variants of 3SAT are *Not–All–Equal 3-Satisfiability* (NAE–3SAT for short) and *1–in–3 Satisfiability* (1–IN–3SAT). Both these problems are defined on formulae in which each clause has size exactly three. A truth assignment t is *NAE satisfying* if each clause has at least one true and at least one false literal, and t is called *1–in–3 satisfying* if each clause has exactly one true literal. The notions of *NAE satisfiable* and *1–in–3 satisfiable* formulae, and the corresponding decision problems are defined in the obvious way. SAT is one of the basic $\mathcal{NP}$-complete problems, and it is well known that NAE–3SAT and 1–IN–3SAT are $\mathcal{NP}$-complete even for formulae without negations [5].

We say that a formula φ for which all clauses have five literals is *2–or–3–in–5 satisfiable* if there exists a truth assignment such that in each clause either two or three literals are true. Let 2–OR–3–IN–5SAT denote the decision problem asking whether a given formula without negations is 2–or–3–in–5 satisfiable. For a formula φ, in which all clauses have six literals, a truth assignment t is *3–in–6 satisfying* if each clause in φ has exactly three true literals. The formula φ is *3–in–6 satisfiable* if there exists a 3–in–6 satisfying truth assignment. 3–IN–6SAT is the decision problem asking whether a given formula φ is 3–in–6 satisfiable. The fact that 2–OR–3–IN–5SAT is $\mathcal{NP}$-complete and that 3–IN–6SAT is $\mathcal{NP}$-complete for formulae without negations follows from [5].

Now we strengthen the result about 3–IN–6SAT.

Proposition 1. *Problem* 3–IN–6 SAT *is* $\mathcal{NP}$*-complete for planar formulae without negations.*

Proof. Suppose we have a formula φ with clauses of size six without negations. We now show that if the formula φ is not planar we can alter it in polynomial time so that the resulting formula φ' is planar and φ is 3–in–6 satisfiable if and only if φ' is 3–in–6 satisfiable. This will prove the lemma. Let d be a drawing

of the incidence graph of φ in the plane, such that any two edges cross at most once. For each pair of crossing edges $e = (v, c)$ and $e' = (v', c')$, add four new variables $v_1^{ee'}, \ldots, v_4^{ee'}$ and three clauses $c^{ee'} = v \vee v \vee v_1^{ee'} \vee v_1^{ee'} \vee v' \vee v_2^{ee'}, c_e^{ee'} = v_1^{ee'} \vee v_1^{ee'} \vee v_1^{ee'} \vee v_3^{ee'} \vee v_3^{ee'} \vee v_3^{ee'}, c_{e'}^{ee'} = v_2^{ee'} \vee v_2^{ee'} \vee v_2^{ee'} \vee v_4^{ee'} \vee v_4^{ee'} \vee v_4^{ee'}$. Then substitute occurrences of v in c by $v_3^{ee'}$, and occurrences of v' in c' by $v_4^{ee'}$. See Figure 1 for an example of a gadget for two crossing edges.

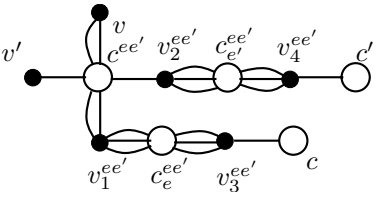


Fig. 1. The crossing gadget for two edges $\{v, c\}$ and $\{v', c'\}$. Empty circles represent clauses, and full circles represent variables

After the substitutions we clearly obtain a planar formula. It remains to prove that φ' is 3–in–6 satisfiable if and only if φ is. To do so, we show that 3–in–6 satisfiability of the formula is unchanged by each substitution. Let t be a 3–in–6 satisfying truth assignment for φ and let ψ be the formula obtained from φ by the substitution described above. Setting $t'(x) = t(x)$ for all variables x of φ and $t'(v_1^{ee'}) = \neg t(v), t'(v_2^{ee'}) = \neg t(v'), t'(v_3^{ee'}) = t(v)$ and $t'(v_4^{ee'}) = t(v')$, we obtain a 3–in–6 satisfying truth assignment for ψ. The other implication can be seen as follows. Let t' be a 3–in–6 satisfying truth assignment for ψ. Hence it must hold that $t'(v_1^{ee'} = \neg t'(v_3^{ee'})$ and $t'(v_2^{ee'}) = \neg t'(v_4^{ee'})$. It is also clear that $t'(v) = \neg t'(v_1^{ee'}) = t'(v_3^{ee'})$. Thus, regardless of the truth assignment, there are two true and two false literals in the clause $c^{ee'}$. Hence $t'(v') = \neg t'(v_2^{ee'}) = t'(v_4^{ee'})$ and we can conclude (because $t'(v) = t(v_3^{ee'})$ and $t'(v') = t(v_4^{ee'})$) that if t' is restricted to the variables of φ, then a 3–in–6 satisfying truth assignment is obtained.

3 $\mathcal{NP}$-Hardness of Balanced Ordering Problems

In this section we prove several $\mathcal{NP}$-hardness results about balanced ordering problems.

Theorem 1. *The* PERFECT ORDERING *problem is $\mathcal{NP}$-complete for graphs with maximum degree four.*

Proof. The construction is a refinement of a construction by Biedl *et al.* [1]; the difference being that we reduce the maximum degree from six to four. $\mathcal{NP}$-hardness is proved by a reduction from NAE–SAT. Given a formula φ, create a graph G_φ with one vertex u_c for each clause c. For each variable v that occurs o_v times in φ, add a path on $3o_v + 1$ new vertices $p_1^v, \ldots, p_{3o_v+1}^v$ to G_φ, add o_v

additional vertices $q_1^v, \ldots, q_{o_v}^v$ and connect $q_i^v, i \in \{1, \ldots, o_v\}$ with vertices p_{3i-2}^v and p_{3i}^v of the path. The path with the additional vertices is called a *variable gadget*. Finally for each $i \in \{1, \ldots, o_v\}$, connect vertex p_{3i-2}^v of the path to u_c, where c is the clause corresponding to the i-th occurrence of the variable v. These edges are called *clause edges*. See Figure 2 for an example of this construction.

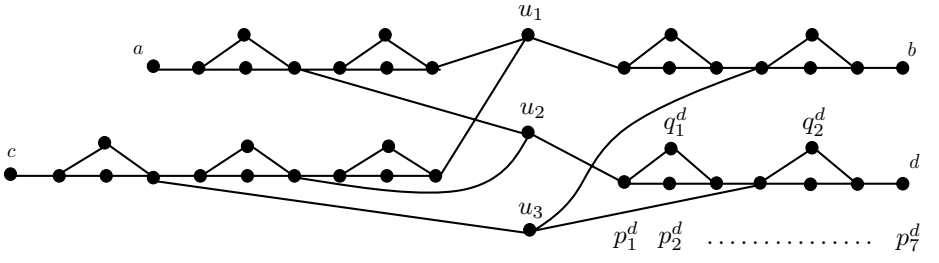


Fig. 2. Constructed graph for formula $(a \vee b \vee c) \wedge (c \vee a \vee d) \wedge (d \vee c \vee b)$. The three clauses have numbers $1, 2, 3$ in the picture

Observe that the maximum degree of G_φ is four. In particular, $\deg(u_c) = 3$, $\deg(q_i^v) = 2$ for all $i \in \{1, \ldots, o_v\}$, $\deg(p_{3i}^v) = 3$ for all $i \in \{1, \ldots, o_v\}$, $\deg(p_{3i-2}^v) = 4$ for all $i \in \{2, \ldots, o_v\}$, $\deg(p_{3i-1}^v) = 2$ for all $i \in \{1, \ldots, o_v\}$, $\deg(p_1^v) = 3$, and $\deg(p_{3o_v+1}^v) = 1$.

We now prove that G_φ has a perfect ordering if and only if φ is NAE-satisfiable. Suppose G_φ has a perfect linear ordering σ. For each variable v, since $\deg(p_{3i-1}^v) = 2$ and $\deg(q_i^v) = 2$, vertices $p_{3i-1}^v, i \in \{1, \ldots, o_v\}$, and $q_i^v, i \in \{1, \ldots, o_v\}$, must have one incident edge to the left and one to the right in σ. Thus they must be placed between p_{3i-2}^v and p_{3i}^v. As p_{3i-1}^v and q_i^v are on one side (e.g., to the left) of vertex p_{3i-2}^v (p_{3i}^v) the other neighbours of the vertex must be on the other side. This implies that in σ, the path in each variable gadget is in the order given by its numbering or inverse numbering, and all the clause edges (the edges with exactly one endpoint in the variable gadget) have a clause vertex on the same end (for example the left end of each clause edge is a vertex of a path). If the path in the gadget for variable v is ordered according to its numbering, then set $t(v) := 0$. Otherwise set $t(v) := 1$. This truth assignment is NAE-satisfying because each clause vertex has at least one neighbour on each side.

For a given truth assignment t we can analogously construct a perfect linear ordering. First place each variable gadget corresponding to a variable with $t(v) = 0$ with the path placed according to the inverse ordering, and put each vertex q_i^v immediately after vertex $p_{3i-1}^v, i \in \{1, \ldots, o_v\}$. Then place vertices u_c in an arbitrary order and finally the variable gadgets corresponding to variables with $t(v) = 1$ with the paths ordered according to the numbering and vertices q_i^v placed immediately after the vertex p_{3i-2}^v. □

Now we present the result about ordering of planar graphs:

Theorem 2. *The* PERFECT ORDERING *problem is $\mathcal{NP}$-complete for planar simple graphs with maximum degree six.*

Proof. We reduce the problem of 3–IN–6 SAT for planar formulae to the PERFECT ORDERING problem for planar graphs. To do so, use the graph construction from the proof of Theorem 1. Note that multiple occurrences of a variable in a clause do not create any parallel edges in the constructed graph. Clearly the construction creates planar graph of maximum degree six from a planar formula and perfect orderings of the created graph correspond to 3–in–6 satisfying truth assignments, as in the proof of Theorem 1. □

The following two technical lemmas will be used later for removing parallel edges from a multigraph without changing an ordering with minimum imbalance. Their proofs are omitted due to the space limitation.

Lemma 1. *Let G be the multigraph drawn in Figure 3 with two parallel edges added between the vertices a and b. Then there exists a minimum ordering of G such that a is the leftmost and b the rightmost vertex. Such an ordering is called a* natural ordering *of G.*

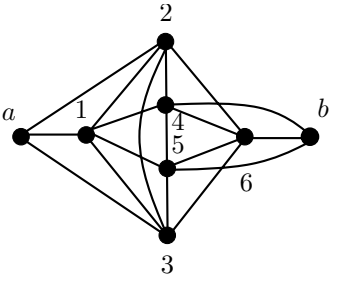


Fig. 3. The triple edge gadget

Lemma 2. *Let G be a 5-regular multigraph and let c be the number of triple-edges in G. Let G' be the graph obtained from G by replacing each triple-edge of G with endpoints a and b by the* triple-edge gadget *in Figure 3. The vertices a and b of the gadget are identified with the original end-vertices of the triple-edge. Then $M(G) = M(G') - 10 \cdot c$.*

For the next reduction we use the 2–OR–3–IN-5SAT problem which we proved to be $\mathcal{NP}$-complete in Section 2.

Theorem 3. *The* PERFECT ORDERING *problem is $\mathcal{NP}$-complete for 5-regular multigraphs.*

Proof. We prove $\mathcal{NP}$-hardness by a reduction from 2–OR–3–IN-5SAT. Suppose that we are given a formula φ without negations and with all clauses of size five. Moreover we assume that each variable occurs in at least two different clauses in the formula. We can make a formula satisfy this condition by adding satisfied clauses of type $x \vee x \vee x \vee \neg x \vee \neg x$. Now create the following multigraph G from

φ. For each clause c add a new vertex v_c to G. For each variable x that occurs o_x times in φ, add a new path (called a *variable path*) with $2o_x - 2$ vertices $v_1^x, \ldots, v_{2o_x-2}^x$ where edges $v_{2i-1}^x v_{2i}^x, 1 \leq i \leq o_x - 1$, are triple-edges. Connect vertex $v_{2i}^x, 1 \leq i \leq o_x - 1$, of the path to the vertex corresponding to the clause with i-th occurrence of x. Furthermore, connect vertex $v_{2o_x-2}^x$ to the vertex corresponding to the clause with the o_x-th occurrence of x (because x was in at least two different clauses we can without loss of generality assume that no parallel edges are created). Connect each vertex $v_{2i-1}^x, 1 \leq i \leq o_x - 1$, to the new vertex p_i^x, and connect each vertex v_1^x to the new vertex p_0^x. Now the only vertices which have degree other that five are in the set $P = \{p_j^x : x \text{ is a variable}, 0 \leq j \leq o_x - 1\}$ and these have degree one. By running the following procedure two times for the set P, all the vertices will have degree five.

$n := |P|$
Arbitrarily number the vertices in P by $1, \ldots, n$.
`while` $|P| \geq 3$ `do`
  Take three arbitrary vertices $u_i, u_j, u_k \in P$
  $P := P \setminus \{u_i, u_j, u_k\} \cup \{u_{n+1}, u_{n+2}\}$
  Add a complete bipartite graph on u_i, u_j, u_k and u_{n+1}, u_{n+2} to G.
  $n := n + 2$
`end`
Now $P = \{u_i, u_j\}$
Add to G a complete bipartite graph on u_i, u_j and new vertices s_1, s_2.
Add a triple-edge $s_1 s_2$ to G.

Let n_0 denote the value of n at the beginning of the procedure and n_1 the value of n at the end of the procedure. It is easy to check that G is 5-regular and we show that G has a perfect ordering if and only if φ was 2-or-3-in-5 satisfiable. Suppose we have a perfect ordering of G. In every ordering of s_1, s_2 and their neighbours u_i, u_j, $B(s_1) + B(s_2) > 2$. Thus (from the perfectness of the ordering) the ordering begins s_1, s_2 without loss of generality. By a similar argument, the ordering ends by vertices s_2', s_1', where s_1' and s_2' are the vertices added in the end of the second run of the procedure on P. Because all other vertices must be balanced we know that every variable path is either in its natural ordering or reversed. Moreover all edges between the variable path and clauses have clause vertices to the right (or to the left in the reversed case). Because all clause vertices are balanced we get a 2-or-3-in-5 satisfying truth assignment of φ by assigning $t(x) = 0$ to the variables whose path is in natural order and $t(x) = 1$ to the variables whose path is reversed. For the other implication, suppose we have a 2-or-3-in-5 satisfying truth assignment t of φ. We can place vertices $s_1, s_2, u_{n_1}, \ldots, u_{n_0+1}$ added in the first run, then vertices $p_j^x : x$ is a variable with $t(x) = 0, 0 \leq j \leq o_x - 1$, then variable paths for variables x such that $t(x) = 0$ in their natural ordering, then the clause vertices, and then symmetrically the rest of the paths and vertices added in the second run. It is straightforward to check that this ordering is perfect. □

See an example of the above construction in Figure 4.

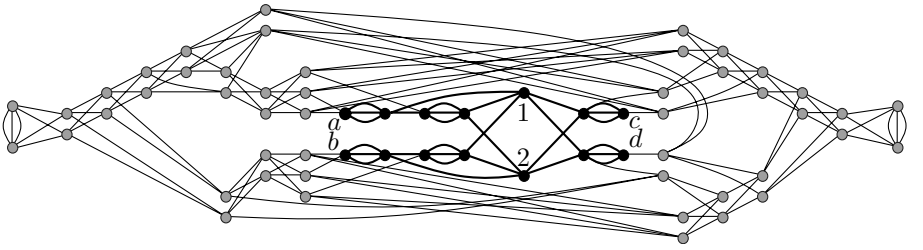


Fig. 4. Constructed 5-regular multigraph for formula $(a \vee a \vee b \vee c \vee d) \wedge (a \vee b \vee b \vee c \vee d)$. Clause vertices are marked 1 and 2. Clause vertices and variable paths are drawn in black colour, vertices p_i^x and vertices added by the procedure are in gray colour

Corollary 1. *Finding a minimum ordering for 5-regular graphs is $\mathcal{NP}$-hard.*

Proof. Construct the multigraph G as in the reduction in the proof of Theorem 3. Say G has c triple edges. Construct G' from G by substituting each triple-edge by a triple-edge gadget. Observe that G' remains 5-regular and is a simple graph. From Lemma 2 we know that orderings of G' with imbalance $|V| + 10 \cdot c$ correspond to perfect orderings of G. This proves $\mathcal{NP}$-hardness of finding the ordering with such imbalance and hence the statement of the corollary. □

4 Algorithm

In this section we present an algorithm that determines in polynomial time whether a given multigraph G has an ordering with an imbalance smaller than a fixed constant. First we introduce a key lemma.

Lemma 3. *There is an $O(n + m)$ time algorithm to test whether a multigraph G with n vertices and m edges has an ordering σ in which a given list of vertices $imbalanced = (v_1, \ldots, v_k)$ are the only imbalanced vertices, and $\sigma(v_i) < \sigma(v_{i+1})$ for all $1 \leq i \leq k - 1$.*

Proof. The vertices not in the list *imbalanced* are called *balanced*. The algorithm works as follows: First we check that all odd-degree vertices are in the *imbalanced* list. If not, then we can reject since every odd-degree vertex must be imbalanced. Now assume that all balanced vertices have even degrees. Then start building an ordering σ from left to right. We append to σ vertices that have not been placed yet and have half of their neighbours already placed. Such vertices are called *saturated* and are stored in the set *saturated*. Because saturated vertices are balanced each saturated vertex must be placed before any of its unplaced neighbours. In particular saturated vertices must form an independent set. Hence we cannot make a mistake when placing any saturated vertices. If there is no saturated vertex, the vertex which is placed next will be imbalanced and hence it must be the first unused vertex from the *imbalanced* list. It remains to prove that it is not better to place some vertices from the *imbalanced* list while there are still some saturated vertices. If the order of vertices of any edge does not

change then we have an equivalent ordering. Otherwise it does change, in which case some balanced vertex becomes imbalanced (as the order of vertices in an edge can change only for the edges which contain at least one balanced vertex) and we would not get a valid ordering. □

The following theorem is a consequence of Lemma 3.

Theorem 4. *There is an algorithm that, given an n-vertex m-edge multigraph G, computes a minimum ordering of G with at most k imbalanced vertices (or answers that there is no such ordering) in time $O(n^k \cdot (m+n))$.*

Proof. The algorithm is simple: just try all the possible choices of k imbalanced vertices and their orderings; for each choice run the procedure from Lemma 3 and select the ordering with minimum imbalance from those orderings. There are $O(n^k)$ k-tuples of imbalanced vertices, and for each such k-tuple, by Lemma 3, we can check in $O(m+n)$ time whether there is an ordering with the chosen vertices imbalanced, and compute the imbalance of the ordering in the case the procedure produced one. □

Corollary 2. *There is a polynomial time algorithm to determine whether a multigraph G has an ordering with imbalance less than a fixed constant c.*

Proof. Apply the algorithm from Theorem 4 with $k = c - 1$. If the algorithm rejects the multigraph or produces an ordering with imbalance greater than c, then the graph does not have an ordering with imbalance less than c (because any ordering with imbalance less than c must have at most $c - 1$ imbalanced vertices). If the algorithm outputs some ordering with imbalance less than c, then we are also done. □

Corollary 3. *The* PERFECT ORDERING *problem is solvable in time $O(n^2(n + m))$ for any n-vertex m-edge multigraph with all vertices of even degree.*

Proof. Apply the algorithm from Theorem 4 with $k = 2$, and then check whether the achieved imbalance is equal to that required by the PERFECT ORDERING problem. A perfect ordering of a multigraph with even degrees must have exactly two imbalanced vertices (if there is at least one edge). □

5 Conclusion and Open Problems

In this paper we have considered the problems of deciding the existence of a perfect ordering for graphs with maximum degree four, planar graphs with maximum degree six and 5-regular multigraphs. All these problems were shown to be $\mathcal{NP}$-complete, thus answering a number of questions raised by Biedl *et al.* [1]. The result for planar graphs still leaves unresolved the complexity of the PERFECT ORDERING problem for planar graphs with maximum degree four or five. We have also established that it is $\mathcal{NP}$-hard to find an ordering with minimum imbalance for 5-regular simple graphs. In the positive direction, we have presented an algorithm for determining an ordering with imbalance smaller than

k running in time $O(n^k(n+m))$. It would be interesting to obtain a fixed-parameter-tractable (FPT) algorithm for this problem (as one cannot hope for a polynomial solution unless $\mathcal{P} = \mathcal{NP}$).

References

1. Therese Biedl, Timothy Chan, Yashar Ganjali, MohammadTaghi Hajiaghayi, David R. Wood, Balanced vertex-orderings of graphs, *Discrete Applied Mathematics* 148(1), pp. 27–48, 2005.
2. Goos Kant, Drawing planar graphs using the canonical ordering, *Algorithmica* 16, pp. 4–32, 1996.
3. Goos Kant and Xin He, Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems, *Theoretical Computer Science* 172(1–2), pp. 175–193, 1997.
4. Achilleas Papakostas and Ioannis G. Tollis, Algorithms for area-efficient orthogonal drawings, *Computational Geometry: Theory and Applications* 9, pp. 83–110, 1998.
5. Thomas J. Schaefer, The complexity of satisfiability problems, *Proceedings of 10th Annual ACM Symposium on Theory of Computing* (STOC'78), pp. 216–226, ACM, 1978.
6. David R. Wood, Minimizing the number of bends and volume in three-dimensional orthogonal graph drawings with a diagonal vertex layout, *Algorithmica* 39, pp. 235–253, 2004.
7. David R. Wood, Optimal three-dimensional orthogonal graph drawing in the general position model, *Theoretical Computer Science* 299 (1–3), pp. 151-178, 2003.

An $O(2^{O(k)}n^3)$ FPT Algorithm for the Undirected Feedback Vertex Set Problem*

Frank Dehne[1], Michael Fellows[2], Michael A. Langston[3], Frances Rosamond[2], and Kim Stevens[4]

[1] Carleton University, Ottawa, Canada
frank@dehne.net
[2] University of Newcastle, Callaghan NSW 2308, Australia
{mfellows,fran}@cs.newcastle.edu.au
[3] University of Tennessee, Knoxville TN 37996-3450, USA
langston@cs.utk.edu
[4] The Mechanics Institute, Bob's Farm NSW 2316, Australia
wonganellawines@telstra.com

Abstract. We describe an algorithm for the Feedback Vertex Set problem on undirected graphs, parameterized by the size k of the feedback vertex set, that runs in time $O(c^k n^3)$ where $c = 10.567$ and n is the number of vertices in the graph. The best previous algorithms were based on the method of bounded search trees, branching on short cycles. The best previous running time of an FPT algorithm for this problem, due to Raman, Saurabh and Subramanian, has a parameter function of the form $2^{O(k \log k / \log\log k)}$. Whether an exponentially linear in k FPT algorithm for this problem is possible has been previously noted as a significant challenge. Our algorithm is based on the new FPT technique of iterative compression. Our result holds for a more general "annotated" form of the problem, where a subset of the vertices may be marked as *not* to belong to the feedback set. We also establish "exponential optimality" for our algorithm by proving that no FPT algorithm with a parameter function of the form $O(2^{o(k)})$ is possible, unless there is an unlikely collapse of parameterized complexity classes, namely FPT = $M[1]$.

1 Introduction

The Feedback Vertex Set problem for undirected graphs can be informally described as the problem of finding a set of vertices that "covers all the cycles" in a graph in the sense that every cycle in the graph includes at least one vertex of a solution set. We consider here a generalization of the problem, where the vertices of the input graph may be annotated according to whether or not they are allowed to belong to a solution set. This generalized form of the problem is formally defined as follows:

* This research has been supported in part by the U.S. National Science Foundation under grant CCR–0075792, by the U.S. Office of Naval Research under grant N00014–01–1–0608, by the U.S. Department of Energy under contract DE–AC05–00OR22725, by the Australian Research Council and by the Australian Centre for Bioinformatics

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 859–869, 2005.

FEEDBACK VERTEX SET (FVS)

Instance: An undirected graph $G = (V, E)$ (loops and multiple edges are allowed), an *annotated* subset $U \subseteq V$ of vertices, and a positive integer k.
Parameter: k
Question: Is there a subset S of the vertices not in U, $S \subseteq V - U$, of size at most k, $|S| \leq k$, such that $G - S$ is acyclic?

The FEEDBACK VERTEX SET problem is NP-complete for both directed and undirected graphs [GJ79]. There are numerous applications of the problem in areas such as circuit testing, deadlock resolution, analyzing manufacturing processes and computational biology [BGNR98, ENSS98, FHS03, FHPSS04, KW90]. The minimization version of the problem is approximable within a factor of 2 in polynomial time [BBF99].

The FVS problem has been extensively studied from the parameterized point of view [BBG00, Bod94, DF92, DF99, KPS04, RSS02, RSS05]. A parameterized problem is said to be *fixed-parameter tractable* (FPT) if it can be solved in time $f(k)n^c$ for some function f (unrestricted), where n is the total input size, k is the declared parameter and c is a constant independent of k and n. This running time may be written as $O^*(f(k))$ in the notation introduced by Woeginger [Woe03] that focuses attention on the exponential time costs due to the parameter and ignores the polynomial time costs due to the overall input size. Highlights of previous research on the FVS problem in the parameterized framework include:
• A randomized FPT algorithm due to Becker et al. [BBG00] running in time $O^*(4^k)$ finds a minimum feedback vertex set of size k with probability at least $1 - (1 - 4^{-k})^{c4^k}$ for an arbitrary constant c.
• After several rounds of improvement, the best previous deterministic FPT algorithm, due to Raman, Saurabh and Subramanian [RSS05], refining some ideas from [RSS02] and [KPS04], has a running time of $O^*(2^{O(k \lg k / \lg \lg k)})$. The basic idea for this and most previous algorithms is to branch on short cycles in a bounded search tree approach. See [DF99, Nie02, Nie05] for surveys of this and other FPT techniques.

A number of problems concerning FVS have notably remained open:

(1) Is there an $O^*(2^{O(k)})$ FPT algorithm for FVS on undirected graphs?

(2) Is there a polynomial-time algorithm that kernelizes FVS on undirected graphs to a kernel of size polynomial in k? See [DF99, Nie02, Nie05] for a discussion of kernelization and FPT.

(3) Is the FVS problem in FPT for directed graphs?

In this paper we answer the first of these significant open problems by an approach based on the relatively new technique of *iterative compression* [RSV04, DFRS04, Ma04, GGHNW05]. As we prepare the final version of this paper, we have become aware that independently a solution to (1) has been described by Guo, et al. [GGHNW05], also based on iterative compression. Our algorithm differs in some details, and has a run time analysis that is superior to

the apparently slightly earlier solution to question (1) described in the upcoming conference paper [GGHNW05].

In the next section we provide a brief discussion of the iterative compression technique and its application to the FVS problem. In §3 we describe our FPT algorithm for the solution-compression form of the FVS problem. In §4 we prove an "optimality" result for our algorithm (giving a lower bound on the possibility of further qualitative improvements). In §5 we conclude with a review of open problems.

2 Iterative Compression Applied to FVS

The FPT technique of *iterative compression* seems first to have appeared in an FPT algorithm devised by Reed, Smith and Vetta for the problem of deleting k vertices to render a graph bipartite [RSV04]. The approach was articulated as a general FPT design technique in [DFRS04]. Some applications of the method can be found in [RSV04, DFRS04, Ma04, GGHNW05].

Here we use this approach to solve the FVS decision problem by recursively solving the following constructive *solution-compression* form of the problem:

Solution Compression for Feedback Vertex Set
Instance: An undirected graph $G = (V, E)$
(loops and multiple edges are allowed),
an annotated subset $U \subseteq V$ of vertices,
a *solution set* $S \subseteq V - U$ such that $G - S$ is acyclic,
where $|S| = k + 1$.
Parameter: k
Output: Either: (1) a solution set S' of size k, or
(2) NO (i.e., no solution of size k is possible).

We employ an FPT algorithm for the above compression form of the FVS problem in the following way. We recursively solve a constructive form of the problem of deciding whether a graph $G = (V, E)$ admits a feedback vertex set of size k with vertices to be chosen from $V - U$. In this constructive form of the decision problem we are required either to produce a solution of size k, if one exists, or to return NO otherwise.

Given an instance $(G = (V, E), U \subseteq V, k)$, we recursively address the constructive decision problem for the instance $(G - v, U, k)$ where v is an arbitrarily chosen vertex in $V - U$. If this recursive call on $G - v$ returns NO, that is, no k-vertex solution for $G - v$ is possible, then clearly the correct answer for G is NO as well.

Alternatively, if the recursive call on the instance $(G - v, U, k)$ returns a k-element solution $S \subseteq V - U$, then $S \cup \{v\}$ is a solution of size $k + 1$ for G. We now employ as a subroutine the FPT algorithm for the solution compression problem. If $f(k)n^c$ is the running time for Solution Compression for FVS, then our recursive solution to the constructive decision problem runs in time $f(k)n^{c+1}$, where n is the number of vertices in the graph G.

3 An FPT Algorithm for FVS Solution Compression

We will use the following *reduction rules* that can be easily applied to simplify (or summarily decide) an instance of the problem. Recall that some vertices (the vertices in U in the problem definition) may be annotated as *not* to belong to a solution set.

Rule 1: The Degree One Rule. If v is a vertex (annotated or not) of degree 1 in G, then delete v and adjust the rest of the input data accordingly.

Rule 2: The Degree Two Rule. If v is a vertex (annotated or not) of degree 2 in G, with neighbors a and b (allowing possibly $a = b$), then modify G by replacing v and its two incident edges with a single edge between a and b (or a loop on $a = b$) and adjust the rest of the input data accordingly.

Rule 3: Annotation Contraction. If u and v are adjacent annotated vertices (that is, $u, v \in U$) then contract one of the edges between u and v and adjust the rest of the input data accordingly.

Rule 4: The Loop Rules. If there is a loop on an annotated vertex v then answer NO. If there is a loop on an unannotated vertex $v \in V - U$ then take v into the solution set, and reduce to the instance $(G - v, U, k - 1)$.

Rule 5: Multiedge Reduction. If there are more than two edges between u and v (annotated or not) then delete all but two of these.

Rule 6: Multiedge Selection. If there is an annotated vertex u that is connected by two edges to an unannotated vertex v, then take v into the solution set, that is, reduce to the instance $(G - v, U, k - 1)$.

The soundness of all these reduction rules is self-evident. In time $O(n)$ we can determine if any of the above reduction rules can be applied to a problem instance. Note that applications of the rules may cascade. We say that an instance is *reduced* if none of the reduction rules can be applied.

Note that if we reduce an instance (G, U, k) to an instance (G', U', k') by a series of applications of the above reduction rules, then given a solution S' of size k' for G', we can in time $O(n)$ recover a solution S of size k for G. We will always assume that the instance we are working with is reduced.

Algorithm for Solution Compression for FVS

Input: A reduced instance $(G = (V, E), U \subseteq V, k)$, and a solution $S \subseteq V - U$ of size $k + 1$.

Output: Either a solution of size at most k, or NO if none exists.

Step 1: Branch on all $2^{k+1} - 1$ subsets of S of size at most k. The branch corresponding to a subset $A \subseteq S$ represents the search for a size k solution S' that includes the vertices of A, that is, $A \subseteq S'$, and that does not include any of the vertices of $S - A = A'$.

Thus, in the instance (G', U', k') that represents this branch of Step 1:
(1) the vertices of A are deleted,
(2) the vertices of A' are annotated,
(3) $k' = k - |A|$, and
(4) the instance is further reduced according to Reduction Rules (1-6).

We will argue below that for the reduced instance $(G' = (V', E'), U', k')$ considered on any of the branches of Step 1, we have either:
(i) $|V' - U'| \leq 4k$, or
(ii) we can immediately determine that the answer is NO.

Step 2: On each branch of Step 1, exhaustively analyze the resulting reduced instance by checking each k'-element subset of the unannotated vertices to see if any provides a solution.

Step 2 requires checking at most $\binom{4k}{k}$ subsets. A simple bound on the running time of our algorithm is $O(c^k n^2)$ where $c = 18.963$, since

$$\binom{4k}{k} \approx (9.4815)^k$$

A more refined version of our algorithm, detailed in §3.3, runs in time $O^*(10.567^k)$.

3.1 The Reduced Instance Bound for Step 1

The correctness of the algorithm is obvious because of its extreme simplicity. What is less obvious is the claimed bound of $4k$ on the number of unannotated vertices in the reduced instance generated on a branch of Step 1 that need to be considered further.

Let $A \subseteq S$ and $A' = S - A$ as in the description of Step 1. The immediate instance graph G' on the A-branch of Step 1 consists of two sets of vertices:
(1) The (now) annotated vertices of A', where we have the bound $|A'| \leq k + 1$.
(2) The other vertices, which we denote F. Some of these may be annotated.

This immediate branch instance is further reduced, and this reduction process may result in some modification of the above picture. For example, connected components of the subgraph generated by A' would be contracted to a single vertex, by repeated applications of Rule 3. To simplify the argument, we will assume that the immediate branch instance is already reduced so that our description of the vertices of G' as partitioned into A' and F is accurate (these sets would be modified by further reduction, but a bipartition with the same properties we make use of below would result in any case). The following structural claims hold.

Lemma 1. *The subgraph $\langle F \rangle$ induced by F is acyclic.*

Proof. Otherwise S would not be a solution for G.

Henceforth we may use F (for convenience) to denote also the forest induced by the vertices in the vertex set F.

Lemma 2. *Each leaf l of the forest F is adjacent to at least two distinct vertices in A'.*

Proof. In view of Lemma 1 and Reduction Rules 1 and 2, there must be at least two edges connecting l to vertices in A'. Reduction Rule 6 would apply if l were connected to only one vertex of A'.

The vertices in the forest F can be partitioned into three sets. Let L denote the leaves of F, let J be the vertices that have degree 2 in the forest subgraph $\langle F \rangle$. We will refer to the vertices of J as the *subdivision vertices* of F. Let B, the *branch vertices* of F, be the vertices of degree at least 3 in the subgraph $\langle F \rangle$.

Lemma 3. *Each vertex $j \in J$ is connected to at least one vertex of A'.*

Proof. Otherwise, in view of Lemma 1, Reduction Rule 2 would apply.

Definition 1. *Let F be a forest with vertex set partitioned into the three sets: (1) the leaves L, (2) the subdivision vertices J, and (3) the branch vertices B of F. A* path-matching *of the J-vertices of F of size r consists of:*
(1) r mutually disjoint 2-element subsets $\{x_i, y_i\} \subseteq J$, $1 \leq i \leq r$,
(2) for each i, $1 \leq i \leq r$, a path ρ_i in F from x_i to y_i, subject to the requirement that for $i \neq j$, the paths ρ_i and ρ_j are vertex disjoint.

Definition 2. *The* potential $\pi(F)$ *of the forest F is defined to be the sum of the number of leaves $|L|$ of F and the size of a maximum path-matching of the J-vertices. (See Figure 1 for an example.)*

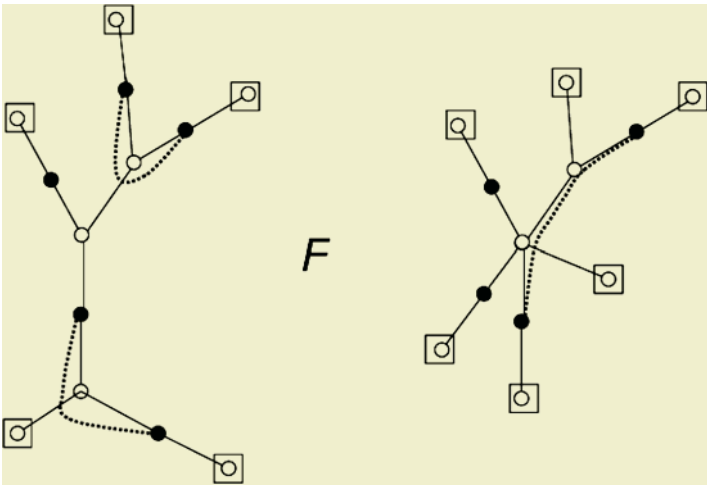


Fig. 1. A maximum path-matching of the subdivision vertices ("J vertices") of the forest F, showing that $\pi(F) = 11 + 3 = 14$

Lemma 4. *Suppose that for the reduced instance (G', U', k') with vertex set partitioned into A' and F as above we have $\pi(F) \geq k' + |A'|$. Then the answer for this instance is NO.*

Proof. If it were a YES-instance (for k') then there would be a feedback vertex set S' consisting of at most k' unannotated vertices. But then there would necessarily be at least $|A'|$ leaves and $J - matching$ paths ρ_i in F having empty intersection with S'. Since $S' \cap A' = \emptyset$ (because the vertices of A' are annotated), there are

at least $|A'|$ *virtual edges* or *virtual loops* connecting the vertices of A' through $F - S'$. (For example, if a leaf l of F is not in S', then by Lemma 2 it is adjacent to two vertices a and b in A', which we consider here as a *virtual edge* between a and b. If the path ρ_i in F from the J-vertex x_i to the J-vertex y_i does not contain any vertices in S', then together with the connections of x_i and y_i to the set A' guaranteed by Lemma 3, we have what can be considered either a *virtual edge* between A' vertices — or a *virtual loop*, in case the A'-adjacencies guaranteed for x_i and y_i by Lemma 3 connect these vertices to the *same* vertex of A'.) Joining the vertices of A' by $|A'|$ virtual edges or virtual loops necessarily implies that there is a cycle not including any vertices of S', that is, that S' is not a feedback vertex set, a contradiction.

Lemma 5. *For any forest F on m vertices, $\pi(F) \geq (m+1)/2$.*

The proof of Lemma 5 is intricate, and can be found in the full paper.

Lemma 6. *If on the branch of Step 1 corresponding to $A \subseteq S$ we have a reduced instance (G', U', k') where the vertices of G' are partitioned into A' and F as in the discussion above, and where $|F| \geq 4k+1$, then this is a NO-instance.*

Proof. By Lemma 5, $\pi(F) \geq 2k+1$. The rest follows by Lemma 4, since $|A'| \leq k+1$ and $k' \leq k$.

3.2 A More Efficient Version

Lemma 4 shows that there is a simple way to improve the efficiency of our algorithm. On the branch of Step 1 corresponding to a subset A of the $(k+1)$-sized solution S, we can answer NO if for the reduced instance we have $\pi(F) \geq k' + |A'|$. Since $k' = k - |A|$ and $|A'| = k+1-|A|$, and using Lemma 5, the total bound on the number of possible solutions explored in Steps 1 and 2 is

$$\sum_{i=0}^{k} \binom{k+1}{i} \binom{2((k+1-i)+(k-i)-1)-1}{k-i} = \sum_{i=0}^{k} \binom{k+1}{i} \binom{4k-4i-1}{k-i}$$

Define

$$f(x,k) = \binom{k}{x} \binom{4(k-x)}{k-x}$$

and suppose $f(x,k)$ is maximized for $x^* = x(k)$. Then our sum above is bounded by $(k+1) \cdot f(x^*, k+1)$.

We next work out two estimates $x_1(k)$ and $x_2(k)$ such that

$$x_1(k) \leq x^*(k) \leq x_2(k)$$

and we will therefore have a bound on our sum of

$$(k+1) \cdot \binom{k+1}{x_2(k+1)} \binom{4((k+1)-x_1(k+1))}{(k+1)-x_1(k+1)}$$

(The reason for the two estimates is that the first part of $f(x,k)$ increases with x, and the second part decreases with x.)

We study the ratio $f(x,k)/f(x+1,k)$. The maximizing value x^* is located (essentially) at the point where this ratio is equal to 1. Assuming that k is large, this ratio is approximately:

$$\frac{f(x,k)}{f(x+1,k)} \approx \left(\frac{x+1}{k-x}\right)(4)(4/3)^3$$

This yields the estimates:
$x_1(k) = (27/283)k$ and
$x_2(k) = (28/283)k$.

Using the bound (based on Stirling's approximation) that

$$\binom{ak}{bk} \leq \left(\frac{a^a}{b^b(a-b)^{a-b}}\right)^k$$

for constants $a > b$, we obtain the bound on our total cost sum of $(k+1)(10.567)^k$.

4 Optimality

Our FPT algorithm for the problem of Solution Compression for FVS yields, by the approach of §2, an FPT algorithm for the parameterized Feedback Vertex Set problem that runs in time $O(c^k n^3)$ where $c = 10.567$. In qualitative terms, we have given an algorithm with a running time of the form $O^*(2^{O(k)})$. We next show that this is, in a qualitative sense, "optimal" for the problem.

Theorem 1. *There can be no FPT algorithm for* Feedback Vertex Set *with a running time of the form* $O^*(2^{o(k)})$ *unless FPT* $= M[1]$.

Proof. Determining whether a graph on n vertices has a vertex cover of size at most $k \log n$, where the parameter is k, is termed the $k \log n$ Vertex Cover Problem. This "renormalized" form of the well-known FPT Vertex Cover problem is complete for the parameterized complexity class $M[1]$ [DEFPR03, CF04]. The theorem follows because there is a linear-size and parameter-preserving (i.e., $k' = k$) polynomial-time reduction from Vertex Cover to Feedback Vertex Set, by simply replacing each edge of the Vertex Cover instance with a pair of parallel edges. Thus if there were an FPT algorithm for Feedback Vertex Set running in time $O^*(2^{o(s)})$ where s is the size of the feedback vertex set, then we would have an algorithm for the $k \log n$ Vertex Cover Problem running in time $O^*(2^{o(k \log n)})$, but as shown in [CJ03], this is an FPT running time. By the completeness of the $k \log n$ Vertex Cover Problem for $M[1]$ we would have FPT $= M[1]$.

Remark 1. The consequence FPT $= M[1]$ is highly unlikely, since it is known that FPT $= M[1]$ if and only if satisfiability of 3SAT instances on n variables can be decided in time $O^*(2^{o(n)})$. (See [DEFPR03, CF04] for further information and discussion.)

Remark 2. A number of other FPT optimality results have been shown for various problems [DFR03, CJ03]. A notable example is the parameterized Planar Dominating Set problem, for which there is an FPT algorithm with a running time of $O^*(2^{O(\sqrt{k})})$ [ABFKN02]. It has been shown that there can be no FPT algorithm for this problem with a running time of the form $O^*(2^{o(\sqrt{k})})$ unless FPT $= M[1]$ [CJ03].

5 Open Problems

There are two compelling problems concerning FVS that remain unresolved.

• Is the Feedback Vertex Set problem for directed graphs in FPT? This is currently open even for the restriction to planar digraphs.

• Is there a polynomial-time kernelization algorithm for FVS on undirected graphs that reduces an instance (G, k) to (G', k') where $k' \leq k$ and the size of G' is bounded by a polynomial in k?

Perhaps an iterative compression approach similar to the one employed in our main result here might be of use in addressing the FVS problem for digraphs.

The potential practical significance of our algorithm should also be investigated. Our approach to the FVS problem here is a new one. The "flat" parallelism of Step 1 (where there are many branches of the algorithm created "all at once", as contrasted with many branches created by repeated binary branching, as is more typically the case for FPT algorithms) could conceivably be significant for highly parallel implementations.

The reduction rules that we have employed are all local and elementary in character. It could be productive to explore if global "crown type" reduction rules for the problem might be possible, as has turned out to be usefully the case for Vertex Cover [ACFLSS04]. Such reduction rules could be important for addressing the very natural open problem concerning polynomial-size kernelization. Alternatively, perhaps some new lower bound techniques, such as those recently developed in [CFKX05], can be used to show that no polynomial-time polynomial-size many:1 kernelization for FVS is possible.

References

[ABFKN02] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks and R. Niedermeier. Fixed parameter algorithms for Dominating Set and related problems on planar graphs. *Algorithmica* 33 (2002), 461–493.

[ACFLSS04] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters and C. T. Symons. Kernelization algorithms for the vertex cover problem: theory and experiments. *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX)*, New Orleans, January, 2004, ACM/SIAM, *Proc. Applied Mathematics 115*, L. Arge, G. Italiano and R. Sedgewick, eds.

[BBF99] V. Bafna, P. Berman and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics* 12 (1999), 289–297.

[BBG00] A. Becker, R. Bar-Yehuda and D. Geiger. Random algorithms for the loop cutset problem. *Journal of Artificial Intelligence Research* 12 (2000), 219–234.

[BGNR98] R. Bar-Yehuda, D. Geiger, J. Naor and R. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. *SIAM Journal on Computing* 27 (1998), 942–959.

[Bod94] H. Bodlaender. On disjoint cycles. *International Journal of Foundations of Computer Science* 5 (1994), 59–68.

[CF04] Y. Chen and J. Flum. On miniaturized problems in parameterized complexity theory. *Proceedings of the First International Workshop on Parameterized and Exact Computation*, Springer-Verlag, *Lecture Notes in Computer Science* vol. 3162 (2004), 108–120.

[CJ03] L. Cai and D. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences* 67 (2003), 789–807.

[CFKX05] J. Chen, H. Fernau, I. Kanj and G. Xia. Parametric duality and kernelization: lower bounds and upper bounds on kernel size. *The 22nd Symposium on Theoretical Aspects on Computer Science* (STACS 2005), Springer-Verlag, *Lecture Notes in Computer Science* vol. 3404 (2005), 269-280.

[DEFPR03] R. Downey, V. Estivill-Castro, M. Fellows, E. Prieto-Rodriguez and F. Rosamond. Cutting up is hard to do: the complexity of k-cut and related problems. *Electronic Notes in Theoretical Computer Science* 78 (2003), 205–218.

[DF92] R. Downey and M. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium* 87 (1992), 161–187.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.

[DFR03] F. Dehne, M. Fellows and F. Rosamond. An FPT algorithm for set splitting. *Proceedings of the 29th Workshop on Graph Theoretic Concepts in Computer Science (WG 2003)*, Springer-Verlag, *Lecture Notes in Computer Science* 2880 (2003), 180–191.

[DFRS04] F. Dehne, M. Fellows, F. Rosamond and P. Shaw. Greedy localization, iterative compression and modeled crown reductions: new FPT techniques, an improved algorithm for set splitting and a novel $2k$ kernelization for vertex cover. *Proceedings of the First International Workshop on Parameterized and Exact Computation*, Springer-Verlag, *Lecture Notes in Computer Science* vol. 3162 (2004), 271–280.

[ENSS98] G. Even, J. Naor, B. Scheiber and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica* 20 (1998), 151–174.

[FHPSS04] C. Fried, W. Hordijk, S.J. Prohaska, C.R. Stadler and P.F. Stadler. The footprint sorting problem. *J. Chem. Inf. Comput. Sci.* 44 (2004), 332 -338.

[FHS03] M. Fellows, M. Hallett and U. Stege. Analogs and duals of the MAST problem for sequences and trees. *Journal of Algorithms* 49 (2003), 192–216.

[GGHNW05] J. Guo, J. Gramm, F. Hueffner, R. Niedermeier, S. Wernicke. Improved fixed-parameter algorithms for two feedback set problems. *Proceedings of WADS 2005*, Springer-Verlag, Lecture Notes in Computer Science (2005), to appear.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[KPS04] I. Kanj, M. Pelsmajer and M. Schaefer. Parameterized algorithms for feedback vertex set. *Proceedings of the First International Workshop on Parameterized and Exact Computation*, Springer-Verlag, *Lecture Notes in Computer Science* vol. 3162 (2004), 235–247.

[KW90] A. Kunzmann and H. Wunderlich. An analytical approach to the partial scan problem. *Journal of Electronic Testing: Theory and Applications* 1 (1990), 163–174.

[Ma04] D. Marx. Chordal deletion is fixed-parameter tractable. Manuscript, 2004.

[Nie02] R. Niedermeier. *Invitation to fixed-parameter algorithms,* Habilitationschrift, University of Tubingen, 2002. (Electronic file available from R. Niedermeier.)

[Nie05] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Oxford University Press, forthcoming.

[RSS02] V. Raman, S. Saurabh and C. Subramanian. Faster fixed-parameter tractable algorithms for undirected feedback vertex set. In *Proceedings of the 13th Annual International Symposium on Algorithms and Computation*, Springer, *Lecture Notes in Computer Science* vol. 2518 (2002), 241–248.

[RSS05] V. Raman, S. Saurabh and C.R. Subramanian. Faster algorithms for feedback vertex set. In: *Proceedings of the 2nd Brazilian Symposium on Graphs, Algorithms and Combinatorics*, GRACO 2005, April 27-29, 2005, Angra dos Reis (Rio de Janeiro), Brazil. Elsevier, *Electronic Notes in Discrete Mathematics* (2005), to appear.

[RSV04] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters* 32 (2004), 299–301.

[Woe03] G. J. Woeginger. Exact algorithms for NP-hard problems: a survey. *Proceedings of 5th International Workshop on Combinatorial Optimization-Eureka, You Shrink! Papers dedicated to Jack Edmonds,* M. Junger, G. Reinelt, and G. Rinaldi (Festschrift Eds.) Springer-Verlag, *Lecture Notes in Computer Science* 2570 (2003), 184-207.

Approximating the Longest Cycle Problem on Graphs with Bounded Degree

Guantao Chen[1,⋆], Zhicheng Gao[2], Xingxing Yu[3,⋆⋆], and Wenan Zang[4,⋆⋆⋆]

[1] Department of Mathematics & Statistics, Georgia State University, Atlanta, GA 30303
gchen@cs.gsu.edu
[2] Faculty of Business Administration, University of Macau, Macau, China
ZCGao@umac.mo
[3] School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332
yu@math.gatech.edu
[4] Department of Mathematics, University of Hong Kong, Hong Kong, China
wzang@maths.hku.hk

Abstract. In 1993, Jackson and Wormald conjectured that if G is a 3-connected n-vertex graph with maximum degree $d \geq 4$ then G contains a cycle of length $\Omega(n^{\log_{d-1} 2})$, and showed that this bound is best possible if true. In this paper we present an $O(n^3)$ algorithm for finding a cycle of length $\Omega(n^{\log_b 2})$ in G, where $b = \max\{64, 4d+1\}$. Our result substantially improves the best existing bound $\Omega(n^{\log_{2(d-1)^2+1} 2})$.

1 Introduction

Over the past three decades, the longest cycle problem, one of the classical $\mathcal{NP}$-hard problems, has attracted tremendous attention. Despite arduous research efforts, little progress has been made on the general problem. Essentially, there is no known polynomial time algorithm which guarantees an approximation ratio better than $n/\text{polylog}(n)$, and there is no strong inapproximability result that explains this situation. For graphs with a cycle of length k, it was proved by Björklund and T. Husfeldt [1] that one can find in polynomial time a cycle of length $\Omega((\log k)^2/\log\log k)$. Recently, Gabow [5] showed how to find in polynomial time a cycle of superpolylogarithmic length through a given vertex. In [4], Feder and Motwani improved Gabow's result with some additional condition.

Karger, Motwani, and Ramkumar [9] established that unless $\mathcal{P} = \mathcal{NP}$ it is impossible to find, in polynomial time, a path of length $n - n^\epsilon$ in an n-vertex

⋆ Partially supported by NSA grant H98230-04-1-0300
⋆⋆ Partially supported by NSF grant DMS-0245530, NSA grant MDA904-03-1-0052, and RGC grant HKU7056/04P
⋆⋆⋆ Partially supported by RGC grant HKU7056/04P

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 870–884, 2005.

Hamiltonian graph for any $\epsilon < 1$. They conjectured that it is true even for graphs with bounded degree. On the other hand, Feder, Motwani, and Subi [3] showed that there is a polynomial time algorithm for finding a cycle of length at least $n^{(\log_3 2)/2}$ in any 3-connected cubic n-vertex graph. They also proposed to examine the problem on 3-connected graphs with bounded degree. In fact, the work on this special longest cycle problem dates back to 1993 when Jackson and Wormald [8] proved that every 3-connected n-vertex graph with maximum degree at most d has a cycle of length at least $\frac{1}{2}n^{\log_b 2}+1$, with $b = 6d^2$. Recently, Chen, Xu, and Yu [2] gave a cubic algorithm that, given a 3-connected n-vertex graph with maximum degree at most d, finds a cycle of length at least $n^{\log_b 2}$ with $b = 2(d-1)^2+1$. It was conjectured by Jackson and Wormald [8] that for $d \geq 4$ the right value for b should be a linear function of d, more specifically, $d-1$; this bound, if true, would be best possible as shown by a concrete example. The purpose to this paper is to asymptotically prove this conjecture.

Theorem 1. *Let $n \geq 4$ and $d \geq 4$ be integers. Let G be a 3-connected graph on n vertices, and assume that the maximum degree of G is at most d. Then G contains a cycle of length at least $\frac{1}{2}n^{\log_b 2}+2$, where $b = \max\{64, 4d+1\}$.*

We point out that our proof yields an $O(n^3)$ algorithm for finding such a cycle in G. Moreover, we introduce the coefficient $1/2$ in the bound just in order to simplify the induction basis, and the additional constant 2 is also for induction purpose. To establish Theorem (1.1), we shall actually prove three statements simultaneously.

Theorem 2. *Let $n \geq 5$ and $d \geq 4$ be integers, let $b = \max\{64, 4d+1\}$ and $r = \log_b 2$, and let G be a 3-connected graph of order n. The following statements hold.*

(a) *Let $xy \in E(G)$ and $z \in V(G) - \{x, y\}$, and let t denote the number of neighbors of z distinct from x and y. Assume that the maximum degree of G is at most $d+1$, and every vertex of degree $d+1$ (if any) is incident with the edge zx or zy. Then there is a cycle C through xy in $G-z$ such that $|C| \geq \frac{1}{2}(\frac{(d-1)n}{dt})^r + 2$.*

(b) *Suppose the maximum degree of G is at most d. Then, for any distinct $e, f \in E(G)$, there is a cycle C through e and f in G such that $|C| \geq \frac{1}{2}(\frac{n}{d})^r + 3$.*

(c) *Suppose the maximum degree of G is at most d. Then, for any $e \in E(G)$, there is a cycle C through e in G such that $|C| \geq \frac{1}{2}n^r + 3$.*

Clearly, (c) of Theorem 2 implies Theorem 1 when $n \geq 5$, and Theorem 1 is obvious when $n = 4$. Note the condition in (a) about the maximum degree; it is due to the addition of edges in order to maintain 3-connectivity.

To prove Theorem 2, we need to deal with graphs obtained from 3-connected graphs by deleting a vertex (such as $G - z$ in (a)), and such graphs need not be 3-connected. We shall use a result of Tutte [10] and Hopcroft and Tarjan [6] to decompose such graphs into "3-connected components", find long paths through certain 3-connected components, and apply the convexity of the function $x^{\log_b 2}$

to account for the unused 3-connected components. Our result substantially improves that in [2], and this improvement is mainly obtained by exploiting more sophisticated structural descriptions of 3-connected components.

This paper is organized as follows. In Section 2, we shall recall the notation related to the decomposition result of Tutte [10] and Hopcroft and Tarjan [6]. We shall also state and prove several results on paths in certain 3-connected components. In Section 3, we shall exhibit several properties enjoyed by the function $f(x) = x^{\log_b 2}$ and then use them to prove several lemmas concerning paths in 3-connected components. In Sections 4 and 5, we shall show that each of (a), (b) and (c) can be reduced to (a), (b) and/or (c) for smaller graphs. In Section 6, we shall complete the proof of our main result, and outline an $O(n^3)$ algorithm for finding a desired long cycle in a 3-connected graph with bounded degree.

2 Paths in Cycle Chains

For convenience, we briefly recall the notation which is used to describe the decomposition of a 2-connected graph into 3-connected components. A detailed description can be found in [2] and [6].

Let G be a 2-connected graph. We allow multiple edges (and hence, $E(G)$ is a multi-set). We say that $\{a, b\} \subseteq V(G)$ is a *separation pair* in G if there are subgraphs G_1, G_2 of G such that $G_1 \cup G_2 = G$, $V(G_1) \cap V(G_2) = \{a, b\}$, $E(G_1) \cap E(G_2) = \emptyset$, and $|E(G_i)| \geq 2$ for $i = 1, 2$. Let $G_i' := (V(G_i), E(G_i) \cup \{ab\})$ for $i = 1, 2$. Then G_1' and G_2' are called *split graphs* of G with respect to the separation pair $\{a, b\}$, and the new edge ab added to G_i is called a *virtual* edge. It is easy to see that since G is 2-connected, G_i' is 2-connected or G_i' consists of two vertices and at least three multiple edges between them.

Suppose a multigraph is split, and the split graphs are split, and so on, until no more splits are possible. Then each remaining graph is called a *split component*. No split component contains a separation pair, and therefore, each split component must be one of the following: a triangle, a *triple bond* (two vertices with three multiple edges between), or a 3-connected graph.

It is not hard to see that if a split component of a 2-connected graph is 3-connected then it is uniquely determined. It is also easy to see that, for any two split components G_1, G_2 of a 2-connected graph, we have $|V(G_1) \cap V(G_2)| = 0$ or 2, and if $|V(G_1) \cap V(G_2)| = 2$ then either G_1 and G_2 share a virtual edge between vertices in $V(G_1) \cap V(G_2)$ or there is a sequence of triple bonds such that the first shares a virtual edge with G_1, any two consecutive triple bonds in the sequence share a virtual edge, and the last triple bond shares a virtual edge with G_2.

In order to get unique 3-connected components, we merge some triple bonds and to merge some triangles. Let $G_i' = (V_i', E_i')$, $i = 1, 2$, be two split components, both containing a virtual edge ab. Let $G' = (V_1' \cup V_2', (E_1' - \{ab\}) \cup (E_2' - \{ab\}))$. Then, the graph G' is called the *merge graph* of G_1 and G_2. Clearly, a merge of triple bonds gives a graph consisting of two vertices and multiple edges, which

is called a *bond*. Also a merge of triangles gives a cycle, and a merge of cycles gives a cycle.

Let $\mathcal{D}$ denote the set of 3-connected split components of a 2-connected graph G. We merge the other split components of G as follows: the triple bonds are merged as much as possible to give a set of bonds $\mathcal{B}$, and the triangles are merged as much as possible to give a set of cycles $\mathcal{C}$. Then $\mathcal{B} \cup \mathcal{C} \cup \mathcal{D}$ is the set of the *3-connected components* of G. Note that any two 3-connected components either are edge disjoint or share exactly one virtual edge.

Tutte [10] proved that the above decomposition of a 2-connected graph into 3-connected components is unique. Hopcroft and Tarjan [6] gave a linear time algorithm for finding all 3-connected components of a graph.

Theorem 3. *For any 2-connected graph, the 3-connected components are unique and can be found in $O(|V| + |E|)$ time.*

If we define a graph whose vertices are the 3-connected components of G and two vertices are adjacent if the corresponding 3-connected components share a virtual edge, then it is easy to see that such a graph is a tree, called the *block-bond tree* of G. For convenience, 3-connected components that are not bonds are called *3-blocks*. An *extreme* 3-block is a 3-block that contains at most one virtual edge. That is, either it is the only 3-connected component, or it corresponds to a degree one vertex in the block-bond tree.

A *cycle chain* in a 2-connected graph G is a sequence $C_1C_2 \dots C_k$ of 3-blocks of G such that each C_i is a cycle and there exist bonds B_1, B_2, …, B_{k-1} in G such that $C_1B_1C_2B_2 \dots B_{k-1}C_k$ is a path in the block-bond tree of G. For convenience, we sometimes write $H := C_1C_2 \dots C_k$ for a cycle chain, and view H as the graph $\bigcup_{i=1}^{k} C_i$. The following is a direct consequence of the definition of a cycle chain.

Proposition 1. *Let G be a 2-connected graph and let $C_1C_2 \dots C_k$ be a cycle chain in G. Then deleting all virtual edges with both ends in $V(C_i \cap C_{i+1})$, $1 \le i \le k-1$, results in a cycle.*

Proposition 2. *Let G be a 2-connected graph, let $C_1C_2 \dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u, v\} \neq V(C_1) \cap V(C_2)$ when $k \neq 1$, and let $ab \in E(C_k)$ with $\{a, b\} \neq V(C_{k-1}) \cap V(C_k)$ when $k \neq 1$. Then there is a path in $(\bigcup_{i=1}^{k} C_i) - \{v, ab\}$ from u to $\{a, b\}$ and containing $(\bigcup_{i=1}^{k-1} V(C_{i-1} \cap C_i)) - (\{a, b\} \cup \{u, v\})$.*

A similar argument establishes the following result.

Proposition 3. *Let G be a 2-connected graph, let $C_1C_2 \dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u, v\} \neq V(C_1 \cap C_2)$ when $k \neq 1$, and let $x \in V(C_k)$ with $x \notin V(C_{k-1} \cap C_k$ when $k \neq 1$. Then there is a path in $(\bigcup_{i=1}^{k} C_i) - v$ from u to x and containing $(\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - \{v\}$.*

The following two results are Propositions (2.7) and (2.8) in [2], respectively.

Proposition 4. *Let G be a 2-connected graph, let $C_1C_2\dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u,v\} \neq V(C_1 \cap C_2)$ when $k \neq 1$, $ab \in E(C_k)$ with $\{a,b\} \neq V(C_{k-1} \cap C_k)$ when $k \neq 1$, and $cd \in E(\bigcup_{i=1}^{k} C_i) - \{ab\}$. Suppose $ab \neq uv$ when $k = 1$. Then there is a path P in $(\bigcup_{i=1}^{k} C_i) - ab$ from $\{a,b\}$ to $\{c,d\}$ such that $uv \in E(P)$, $cd \notin E(P)$ unless $cd = uv$, and $(\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) \subseteq V(P)$.*

Proposition 5. *Let G be a 2-connected graph, let $C_1C_2\dots C_k$ be a cycle chain in G, let $uv \in E(C_1)$ with $\{u,v\} \neq V(C_1) \cap V(C_2)$ when $k \neq 1$, $x \in V(C_k)$ with $x \notin V(C_{k-1} \cap C_k)$ when $k \neq 1$, and $cd \in E(\bigcup_{i=1}^{k} C_i)$. Then there is a path P in $(\bigcup_{i=1}^{k} C_i)$ from x to $\{c,d\}$ such that $uv \in E(P)$, $cd \notin E(P)$ unless $cd = uv$, and $(\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) \subseteq V(P)$.*

We conclude this section by recalling two graph operations and three lemmas from [2]. Let G be a graph and let e, f be distinct edges of G. An *H-transform* of G *at* $\{e,f\}$ is an operation that subdivides e and f by vertices x and y respectively and then adds the edge xy. Let $x \in V(G)$ such that x is not incident with e. A *T-transform* of G *at* $\{x,e\}$ is an operation that subdivides e with a vertex y and then adds the edge xy. If there is no need to specify e, f, x, we shall simply speak of an H-transform or a T-transform. The following result is Lemma (3.3) in [2].

Lemma 1. *Let $d \geq 3$ be an integer and let G be a 3-connected graph with maximum degree at most d. Let G' be a graph obtained from G by an H-transform or a T-transform. Then G' is a 3-connected graph, the vertex of G involved in the T-transform has degree at most $d+1$, and all other vertices of G' has degree at most d.*

3 Convex Function and Paths in Block Chains

In this section we prove several lemmas concerning the function $x^{\log_b 2}$. These lemmas will then be used in the proof of Theorem 2 to show that it suffices to find long paths in certain 3-blocks in a 2-connected graph. The first of these is Lemma (3.1) in [2].

Lemma 2. *Let $b \geq 4$ be an integer, and let $m \geq n$ be positive integers. Then $m^{\log_b 2} + n^{\log_b 2} \geq (m + (b-1)n)^{\log_b 2}$.*

When m is sufficiently larger than n, we can improve the above result.

Lemma 3. *Let $b \geq 9$ be an integer, and let m and n be positive integers. Suppose $m \geq \frac{b(b-1)}{4}n$. Then $m^{\log_b 2} + n^{\log_b 2} \geq (m + \frac{b(b-1)}{4}n)^{\log_b 2}$.*

When m is not sufficiently larger than n, we have the following complementary result.

Lemma 4. *Let $b \geq 64$ be an integer, and let $m \geq n$ be positive integers. Suppose $m \leq \frac{b(b-1)}{4}n$. Then $m^{\log_b 2} + n^{\log_b 2} \geq (4m)^{\log_b 2}$.*

The observations in the following lemma will be convenient in the proof of Theorem 2.

Lemma 5. *Let m be an integer, $d \geq 3$, and $b \geq d+1$. If $m \geq 4$ then $m \geq \frac{1}{2}m^{\log_b 2}+3$. If $m \geq 3$ then $m > \frac{1}{2}(\frac{m}{d})^{\log_b 2}+2$. If $m \geq 2$ then $m > \frac{1}{2}(\frac{m}{d})^{\log_b 2}+1$.*

Let us now turn to paths in block chains. Let G be a 2-connected graph. A *block chain* in G is a sequence $H_1H_2\dots H_h$ for which (1) each H_i is a cycle chain in G or a 3-connected 3-block of G, (2) for any $1 \leq s \leq h-1$, H_sH_{s+1} is not a cycle chain, and (3) there exist bonds $B_1, B_2, \dots, B_{h-1}$ such that $H_1B_1H_2B_2\dots B_{h-1}H_h$ form a path in the block-bond tree of G (by also including the tree paths corresponding to H_i when H_i is a cycle chain). For convenience, we sometimes write $\mathcal{H} := H_1H_2\dots H_h$ for a block chain and view $\mathcal{H} = \bigcup_{i=1}^{h} H_i$ as a graph.

Let $H_1H_2\dots H_h$ be a block chain and let $V(H_s \cap H_{s+1}) = \{x_s, y_s\}$ for $1 \leq s \leq h-1$. For each $1 \leq s \leq h$, we define $A(H_s)$ as follows. If H_s is 3-connected then $A(H_s) := V(H_s)$. If $H_s = C_1C_2\dots C_k$ is a cycle chain then let $A(H_s) := (\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - (\{x_{s-1}, y_{s-1}\} \cup \{x_s, y_s\})$ when $1 < s < h$, $A(H_s) := \bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})$ when $s = 1 = h$, $A(H_s) := (\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - \{x_s, y_s\}$ when $s = 1 < h$, and $A(H_s) := (\bigcup_{i=1}^{k-1} V(C_i \cap C_{i+1})) - \{x_{s-1}, y_{s-1}\}$ when $1 < s = h$.

For a block chain $\mathcal{H} = H_1H_2\dots H_h$, we write $\sigma(\mathcal{H}) := \sum_{s=1}^{h} |A(H_s)|$ and $|\mathcal{H}| := |\bigcup_{i=1}^{h} V(H_i)|$.

Lemma 6. *Assume Theorem 2 holds for graphs of order $< n$, and let $\mathcal{H} = H_1H_2\dots H_h$ be a block chain in a 2-connected graph such that $|\mathcal{H}| < n$ and the maximum degree of $\mathcal{H}$ is at most d. Let $uv \in E(H_1)$ such that $\{u, v\} \neq V(H_1 \cap H_2)$ and $\{u, v\}$ is not a cut of H_1. Suppose for all $1 \leq j < h$, $|A(H_j)| \leq (d-1)\sum_{i=j+1}^{h} |A(H_i)|$. Then there exists a path P from u to v in $\mathcal{H}$ such that $|E(P)| \geq \frac{1}{2}(\sigma(\mathcal{H}))^r + 2$.*

Lemma 7. *Assume Theorem 2 holds for graphs of order $< n$, and let $\mathcal{H} = H_1H_2\dots H_h$ be a block chain in a 2-connected graph such that $|\mathcal{H}| < n$ and the maximum degree of $\mathcal{H}$ is at most d. Let $uv \in E(H_1)$ such that $\{u, v\} \neq V(H_1 \cap H_2)$ and $\{u, v\}$ is not a cut of H_1. Then there is a path P in $\mathcal{H}$ from u to v such that $|E(P)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{H})}{d})^r + 2$.*

Proof. Let t be minimum such that $|A(H_t)| \geq (d-1)\sum_{i=t+1}^{h} |A(H_i)|$. If no such t exists, let $t = h$. Then $|A(H_t)| \geq \frac{d-1}{d}\sum_{i=t}^{h} |A(H_i)|$. Let $\mathcal{H}' = H_1H_2\dots H_t$. Then $\sigma(\mathcal{H}') \geq \frac{d-1}{d}\sigma(\mathcal{H})$. By the choice of t and by Lemma 6, there is a path P in $\mathcal{H}'$ (and hence in $\mathcal{H}$) from u to v such that $|E(P')| \geq \frac{1}{2}(\sigma(\mathcal{H}'))^r + 2$. □

Lemma 8. *Assume Theorem 2 holds for graphs of order $< n$, and let $\mathcal{H} = H_1H_2\dots H_h$ be a block-chain in a 2-connected graph such that $|V(\mathcal{H})| < n$ and the maximum degree of $\mathcal{H}$ is at most d. When $h = 1$, if H_1 is 3-connected or H_1 is a cycle then let $uv \in E(H_1)$ and $x \in V(H_1) - \{u, v\}$, and if $H_1 = C_1C_2\dots C_k$*

is a cycle chain with $k \geq 2$ then let $uv \in E(C_1)$ with $\{u,v\} \neq V(C_1 \cap C_2)$ and let $x \in V(C_k) - V(C_{k-1} \cap C_k)$. When $h \geq 2$, if H_1 is 3-connected or H_1 is a cycle then let $uv \in E(H_1)$ with $\{u,v\} \neq V(H_1 \cap H_2)$, if $H_1 = C_1C_2 \ldots C_k$ is a cycle chain with $k \geq 2$ and $V(H_1 \cap H_2) = V(C_k \cap H_2)$ then let $uv \in E(C_1)$ with $\{u,v\} \neq V(C_1 \cap C_2)$, if H_h is a cycle or H_h is 3-connected then let $x \in V(H_1) - \{u,v\}$, and if $H_h = C_1C_2 \ldots C_k$ is a cycle chain with $k \geq 2$ and $V(H_{h-1} \cap H_h) = V(H_{h-1} \cap C_1)$ then let $x \in V(C_k) - V(C_{k-1})$. Suppose the degree of x in $\mathcal{H}$ is at most $d-1$. Then there exists a path P in $\mathcal{H} - v$ from u to x such that

(i) $|E(P)| \geq \frac{1}{2}\sum_{i=1}^{h}(\frac{|A(H_i)|}{d})^r + 1 \geq \frac{1}{2}(\frac{\sigma(\mathcal{H})}{d})^r + 1$,
(ii) $|E(P \cap H_i)| \geq \frac{1}{2}(\frac{|A(H_i)|}{d})^r + 1$, *and*
(iii) $|E(P \cap H_i)| \geq \max\{1, |A(H_i)| - 2\}$ *when H_i is a cycle chain.*

Corollary 1. *Assume the same hypothesis of Lemma 8. Then for any $0 \leq t \leq h$ and for any $pq \in E(H_t)$, there exists a path P in $\mathcal{H}$ from x to $\{p,q\}$ such that*

(i) $|E(P)| \geq \frac{1}{2}|H_0|^r + \frac{1}{2}\sum(\frac{|A(H_i)|}{d})^r + \sum\max\{1, |A(H_i)| - 2\} + 1$, *where the first summation is over all 3-connected H_i's ($1 \leq i \leq h$) and the second summation is over all cycle chains H_i ($1 \leq i \leq n$),*
(ii) *if we require $uv \in E(P)$ then $|E(P)| \geq \frac{1}{2}\sum(\frac{|A(H_i)|}{d})^r + \sum\max\{1, |A(H_i)| - 2\} + 1$, where the first summation is over all 3-connected H_i's ($0 \leq i \leq h$) and the second summation is over all cycle chains H_i ($0 \leq i \leq n$), and*
(iii) $|E(P)| \geq \frac{1}{2}\sum_{i=0}^{h}(\frac{|A(H_i)|}{d})^r + 1 \geq (\frac{\sigma(\mathcal{H})}{d})^r + 1$.

4 Cycles Through Two Edges

In this section, we show how to reduce (a) and (b) of Theorem 2 to (a), (b) and/or (c) of Theorem 2. Note that finding a cycle in (a) of Theorem 2 through xy avoiding z is equivalent to finding a cycle through xz, yz of appropriate length. (This justifies the title of this section.)

First, we reduce (a) of Theorem 2 through the following lemma.

Lemma 9. *Let $n \geq 6$ and $d \geq 4$ be integers, let $b = \max\{64, 4d+1\}$ and $r = \log_b 2$, and assume that Theorem 2 holds for graphs with at most $n-1$ vertices. Let G be a 3-connected graph with n vertices, let $xy \in E(G)$ and $z \in V(G) - \{x,y\}$, and let t denote the number of neighbors of z distinct from x and y. Assume the maximum degree of G is at most $d+1$, and every vertex of degree $d+1$ in G (if any) is incident with the edge zx or zy. Then there is a cycle C through xy in $G - z$ such that $|C| \geq \frac{1}{2}(\frac{(d-1)n}{dt})^r + 2$.*

Next, we show how to reduce (b) of Theorem 2.

Lemma 10. *Let $n \geq 6$ and $d \geq 4$ be integers, let $b = \max\{64, 4d+1\}$ and $r = \log_b 2$, and assume that Theorem 2 holds for graphs with at most $n-1$ vertices. Suppose G is a 3-connected graph on n vertices and the maximum degree of G is at most d. Then for any $\{e,f\} \subseteq E(G)$, there is a cycle C through e, f in G such that $|C| \geq \frac{1}{2}(\frac{n}{d})^r + 3$.*

5 Cycles Through One Edge

In this section, we show how to reduce (c) of Theorem 2 to (a), (b), or (c) of Theorem 2 for smaller graphs.

Lemma 11. *Let $n \geq 6$ and $d \geq 4$ be integers, let $b = \max\{64, 4d+1\}$ and $r = \log_b 2$, and assume that Theorem 2 holds for graphs with at most $n-1$ vertices. Let G be a 3-connected graph on n vertices, and assume the maximum degree of G is at most d. Then for any $e \in E(G)$, there is a cycle C through e in G such that $|C| \geq \frac{1}{2}n^r + 3$.*

Proof. Let $e = xy \in E(G)$. If $G - y$ is 3-connected, then let y' be a neighbor of y other than x. Clearly, $G' := (G - y) + xy'$ is a 3-connected graph with maximum degree at most d. Since $5 \leq |G'| < n$, Theorem 2 holds for G'. By (c) of Theorem 2, there is a cycle C' through xy' in G' such that $|C'| \geq \frac{1}{2}(n-1)^r + 3$. Now let $C := (C' - xy') + \{y, xy, yy'\}$. Then C is a cycle through xy in G and by Lemma 2, $|C| = |C'| + 1 \geq \frac{1}{2}(n-1)^r + 1 + 3 \geq \frac{1}{2}n^r + 3$.

Therefore, we may assume that $G - y$ is not 3-connected. Since $G - y$ is 2-connected, we can use Theorem 3 to decompose $G - y$ into 3-connected components.

First, let us consider the case where all 3-blocks of $G - y$ are cycles. Let $\mathcal{L} = L_1 \dots L_\ell$ be a cycle chain in $G - y$ such that (i) $x \in V(L_1)$, (ii) L_ℓ is an extreme 3-block of $G-y$, and (iii) subject to (i) and (ii), $|\mathcal{L}|$ is maximum. It is easy to see that there is some $y' \in V(\mathcal{L}) - \{x\}$ such that $\bigcup_{i=1}^{\ell} L_s$ contains a Hamilton path P from x to y' and G has a path Q from y' to y disjoint from $V(\mathcal{L}) - \{y\}$. Let $C := (P \cup Q) + \{y, xy, yy'\}$. Then $|C| \geq |\mathcal{L}| + 1$. If $G - y = \mathcal{L}$ then $|C| = n \geq \frac{1}{2}n^r + 3$ (since $n \geq 5$). So we may assume $G - y \neq \mathcal{L}$. Write $B := L_1$. Then by (iii), we have $|\mathcal{L}| \geq \frac{(n-1)-|B|}{t-1} + |B| = \frac{n+(t-2)|B|-1}{t-1}$, where t is the number of extreme 3-blocks of $G - y$ distinct from L_1 (because $x \in V(L_1)$ and $xy \in E(G)$). So $n \geq t + 4$ (since $|B| \geq 3$) and $2 \leq t \leq d - 1$ (because $G - y \neq \mathcal{L}$). Then $|C| \geq |\mathcal{L}| + 1 \geq \frac{n+(t-2)|B|-1}{t-1} + 1$. Note that $|C| - 3 \geq \frac{n+(t-2)|B|-1}{t-1} - 2 \geq \frac{n+t-5}{t-1}$ (since $|B| \geq 3$). Using elementary calculus, we can show that $\frac{n+t-5}{t-1} \geq \frac{1}{2}n^r$. Therefore, $|C| \geq \frac{1}{2}n^r + 3$.

Hence, we may assume that not all 3-blocks of $G-y$ are cycles. Let H_0 be a 3-connected 3-block of $G-y$ such that $|H_0|$ is maximum. Let $\mathcal{H} = H_0 H_1 H_2 \dots H_h$ be a block chain in $G - y$ such that $x \in V(H_h) - V(H_{h-1})$, and if $H_h = C_1 \dots C_k$ is a cycle chain with $k \geq 2$ and $V(H_{h-1} \cap H_h) = V(C_1 \cap C_2)$ then $x \in V(C_k) - V(C_{k-1} \cap C_k)$. For $0 \leq i \leq h - 1$, let $V(H_i \cap H_{i+1}) = \{a_i, b_i\}$. Choose $\mathcal{H}$ so that $\sigma(\mathcal{H})$ is maximum.

If $G - y \neq \mathcal{H}$, there is a block chain $\mathcal{L} := L_1 L_2 \dots L_\ell$ in $G - y$ such that $V(\mathcal{H} \cap \mathcal{L}) = V(\mathcal{H} \cap L_1)$ consists of two vertices c_0 and d_0, L_ℓ contains an extreme 3-block of $G - y$, and if $L_1 = C_1 C_2 \dots C_k$ is a cycle chain with $k \geq 2$ and $V(L_1 \cap L_2) = V(C_k \cap H_2)$ then $c_0 d_0 \in E(C_1)$ and $\{c_0, d_0\} \neq V(C_1 \cap C_2)$. Without loss of generality, we may assume that $c_0 d_0 \in E(H_t) - E(H_{t+1})$. For $1 \leq i \leq \ell - 1$, let $V(L_i \cap L_{i+1}) = \{c_i, d_i\}$. If such $\mathcal{L}$ exists, we choose $\mathcal{L}$ such

that $\sigma(\mathcal{L})$ is maximum. Therefore, since the maximum degree of G is at most d, we have

(1) $\sigma(\mathcal{L}) \geq \frac{n-\sigma(\mathcal{H})-1}{d-1}$.

By Corollary 1, there exists a path P in $\mathcal{H}$ from x to $\{c_0, d_0\}$ such that

(2) $|E(P)| \geq \frac{1}{2}|H_0|^r + \frac{1}{2}\sum(\frac{|H_i|}{d})^r + \sum_{\max}\{1, |A(H_i)| - 2\} + 1$, where the first summation is over all 3-connected H_i's and the second is over those H_i's which are cycle chains.

(3) We may asume $\sigma(\mathcal{H}) < \frac{n-1}{4}$, and hence, $\mathcal{L} \neq \emptyset$.

Suppose $\sigma(\mathcal{H}) \geq \frac{n-1}{4}$. Without loss of generality, assume c_0 is an end of the path P in (2). Because $|H_0| \geq |H_i|$ for all 3-conneced H_i's, it follows from Lemma 2 that $|E(P)| \geq \frac{1}{2}(\sigma(\mathcal{H}))^r + 1$. By Lemma 8, there is a path Q in $\mathcal{L} - d_0$ from c_0 to some $y' \in N(y) \cap V(L_\ell)$ such that $|E(P)| \geq \frac{1}{2}(\frac{\sigma(\mathcal{L})}{d})^r + 1$.

Let $C = (P \cup Q) + \{y, yy', yx\}$. Then $|C| = |E(P)| + |E(Q)| + 2 \geq \frac{1}{2}(\sigma(\mathcal{H}))^r + 1 + \frac{1}{2}(\frac{\sigma(\mathcal{L})}{d})^r + 3$. If $\sigma(\mathcal{H}) \leq \frac{b(b-1)}{4}\sigma(\mathcal{L})$, then by Lemma 4, $|C| \geq \frac{1}{2}(4\sigma(\mathcal{H}) + 1)^r + 3 \geq \frac{1}{2}n^r + 3$. So assume $\sigma(\mathcal{H}) \geq \frac{b(b-1)}{4}\sigma(\mathcal{L})$. By Lemma 3 $|C| \geq \frac{1}{2}(\sigma(\mathcal{H}) + 1 + \frac{b(b-1)}{4}\sigma(\mathcal{L}))^r + 3 \geq \frac{1}{2}n^r + 3$ (by (1)).

(4) We may further assume $|H_0| + 4(\sigma(\mathcal{H}) - |H_0| + \sigma(\mathcal{L}) < n$, in particular, $\sigma(\mathcal{L}) < \frac{n-1}{4}$.

Suppose $|H_0| + 4(\sigma(\mathcal{H}) - |H_0| + \sigma(\mathcal{L}) \geq n$. Without loss of generality, assume that the path P in (2) is from x to c_0. By Lemma 8, there is a path Q in $\mathcal{L} - d_0$ from c_0 to some $y' \in N(y) \cap V(L_\ell)$ such that $|E(Q)| \geq \frac{1}{2}\sum_{i=1}^{\ell}(\frac{|A(L_i)|}{d})^r + 1$. Let $C = (P \cup Q) + \{y, yy', yx\}$. Then by (2), $|C| = |E(P)| + |E(Q)| + 2 \geq \frac{1}{2}|H_0|^r + \frac{1}{2}\sum(\frac{|A(H_i)|}{d})^r + \sum \max\{1, |A(H_i)| - 2\} + \frac{1}{2}\sum_{i=1}^{\ell}(\frac{|A(L_i)|}{d})^r + 4$, where the first summation is over all 3-connected H_i's and the second is over those H_i's which are cycle chains. Using Lemma 2 and the fact $(b-1)/d \geq 4$, we have

$$\begin{aligned}
|C| &\geq \frac{1}{2}[|H_0| + (b-1)\sum(\frac{|A(H_i)|}{d}) \\
&\quad +(b-1)\sum \max\{1, |A(H_i)| - 2\}]^r + \sum_{j=1}^{\ell}(\frac{|A(L_j)|}{d})^r + 4 \\
&\geq \frac{1}{2}[|H_0| + 4(\sum_{i=1}^{h}|A(H_i)| + \sum_{j=1}^{\ell}|A(L_j)|)]^r + 4 \\
&> \frac{1}{2}n^r + 3.
\end{aligned}$$

A block chain $\mathcal{M} := M_1M_2 \ldots M_m$ is called an $\mathcal{HL}$-*leg* if M_m contains an extreme 3-block of $G - y$ and $V(\mathcal{M} \cap (\mathcal{H} \cup \mathcal{L}))$ consists of two vertices x_0 and y_0 such that $\{x_0, y_0\} \subseteq V(M_1)$ and $\{x_0, y_0\} \neq V(M_1 \cap M_2)$, and if $M_1 = C_1C_2 \ldots C_k$ is a cycle chain with $k \geq 2$ and $V(C_k \cap M_2) = V(M_1 \cap M_2)$ then $\{x_0, y_0\} \subseteq V(C_1)$ and $\{x_0, y_0\} \neq V(C_1 \cap C_2)$.

(5) We may assume that there is an $\mathcal{HL}$-leg $\mathcal{M} = M_1M_2 \dots M_m$ such that $\sigma(\mathcal{M}) \geq \frac{(n-1)}{4(d-2)}$.

To prove (5), we choose an $\mathcal{HL}$-leg $\mathcal{M}$ such that $\sigma(\mathcal{M})$ is maximum. Because $\sigma(\mathcal{H}) < \frac{n-1}{4}$ (by (3)) and $\sigma(\mathcal{L}) < \frac{n-1}{4}$ (by (4)) and since the maximum degree of G is at most d, $\sigma(\mathcal{M}) + 2 \geq \frac{n-1}{2(d-2)}$. If $n < 8d$ then $\frac{1}{2}n^r + 3 \leq 4$, and one can easily see that (c) of Theorem 2 holds. So assume $n \geq 8d$. Then we see that $\sigma(\mathcal{M}) \geq \frac{(n-1)}{4(d-2)}$.

For an $\mathcal{HL}$-leg $\mathcal{M}$ in (5), let x_0 and y_0 be the vertices in $V(\mathcal{M} \cap (\mathcal{H} \cup \mathcal{L}))$, and let $V(M_i \cap M_{i+1}) = \{x_i, y_i\}$ for $1 \leq i \leq m-1$. Based on the location of $\{x_0, y_0\}$, we consider four cases.

Case 1. $\mathcal{M}$ may be chosen so that $x \notin \{x_0, y_0\} \cap \{c_0, d_0\}$, $\{x_0, y_0\} \neq \{c_0, d_0\}$, and $\{x_0, y_0\} \subseteq V(\mathcal{L})$.

Because $\{x_0, y_0\} \neq \{c_0, d_0\}$, we may assume $\{x_0, y_0\} \subseteq V(L_t)$ with $\{x_0, y_0\} \neq \{c_{t-1}, d_{t-1}\}$. By the choice of $\mathcal{L}$, $\sum_{i=t+1}^{\ell} |A(L_i)| \geq \sigma(\mathcal{M})$. Without loss of generality, we may assume that the path P in (2) is from x to c_0.

Since each L_i is 3-connected or a cycle chain, there exists a path Q in $(\bigcup_{i=1}^{t} L_i) - d_0$ from c_0 to some $z \in \{c_t, d_t\} \cup \{x_0, y_0\}$ such that (a) if $z \in \{c_t, d_t\}$ then $x_0y_0 \in E(Q)$, and $c_td_t \notin E(Q)$ unless $x_0y_0 = c_td_t$, and (b) if $z \in \{x_0, y_0\}$ then $c_td_t \in E(Q)$, and $x_0y_0 \notin E(Q)$ unless $x_0y_0 = c_td_t$.

Suppose $z \in \{c_t, d_t\}$, and assume the notation is chosen so that $z = c_t$. By Lemma 8 there is a path P_1 in $(\bigcup_{i=t+1}^{\ell} L_i) - d_t$ from z to some $y' \in N(y) \cap V(L_\ell)$ such that

$$|E(P_1) \cap L_i| \geq \begin{cases} \frac{1}{2}(\frac{|L_i|}{d})^r + 1, & \text{if } L_i \text{ is 3-connected,} \\ \max\{1, |A(L_i)| - 2\}, & \text{if } L_i \text{ is a cycle chain.} \end{cases}$$

By Lemma 7, let P_2 be a path from x_0 to y_0 in $\mathcal{M}$ such that $|E(P_2)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{M})}{d})^r + 2$. Let C be the cycle obtained from $(P \cup Q \cup P_1) + \{y, yy', yx\}$ by replacing x_0y_0 with P_2. Then

$$\begin{aligned}
|C| &\geq |E(P)| + |E(P_1)| + |E(P_2)| + 2 \\
&\geq \frac{1}{2}[(\sigma(\mathcal{H}))^r + \sum (\frac{|A(L_i)|}{d})^r + \sum \max\{1, |A(L_i)| - 2\} + (\frac{(d-1)\sigma(\mathcal{M})}{d})^r] + 5 \\
&\geq \frac{1}{2}[(\sigma(\mathcal{H}) + \frac{(b-1)\sum_{i=t+1}^{\ell} |A(L_i)|}{d})^r + (\frac{(d-1)\sigma(\mathcal{M})}{d})^r] + 5 \quad \text{(Lemma 2)} \\
&\geq \frac{1}{2}[\sigma(\mathcal{H}) + \sum_{i=t+1}^{\ell} |A(L_i)| + \frac{(b-1)(d-1)\sigma(\mathcal{M})}{d}]^r + 5 \quad \text{(Lemma 2)} \\
&> \frac{1}{2}(4(d-1)\sigma(\mathcal{M}) + 1)^r + 3 \quad \text{(by Lemma 2)} \\
&\geq \frac{1}{2}n^r + 3.
\end{aligned}$$

The fourth inequality is also because $\sum_{i=t+1}^{\ell} |A(L_i)| \geq \sigma(\mathcal{M})$, and the last inequality follows from (5).

Now suppose $z \in \{x_0, y_0\}$, and assume the notation is chosen so that $z = x_0$. By Lemma 8, there is a path P_2 in $\mathcal{M} - y_0$ from x_0 to some $y'' \in N(y) \cap V(M_m)$ such that

$$|E(P_2 \cap M_i)| \geq \begin{cases} \frac{1}{2}(\frac{|M_i|}{d})^r + 1, & \text{if } M_i \text{ is 3-connected,} \\ \max\{1, |A(M_i)| - 2\}, & \text{if } M_i \text{ is a cycle chain.} \end{cases}$$

By Lemma 7 there is a path P_1 in $\bigcup_{i=t+1}^{\ell} L_i$ from c_t to d_t such that $|E(P_1)| \geq \frac{1}{2}(\frac{(d-1)\sum_{i=t+1}^{\ell} |A(L_i)|}{d})^r$. Let C be the cycle obtained from $(P \cup Q \cup P_2) + \{y, yx, yy''\}$ by replacing $c_t d_t$ with P_1. Similarly, we can show that

$$\begin{aligned} |C| &\geq \frac{1}{2}[(\sigma(\mathcal{H}))^r + \sum (\frac{|A(M_i)|}{d})^r + \sum \max\{1, |A(M_i)| - 2\} \\ &\quad + (\frac{(d-1)\sum_{i=t+1}^{\ell} |A(L_i)|}{d})^r] + 5 \\ &\geq \frac{1}{2}[(\sigma(\mathcal{H}) + \sigma(\mathcal{M}))^r + (\frac{(d-1)\sum_{i=t+1}^{\ell} |A(L_i)|}{d})^r] + 5. \end{aligned}$$

If $\sum_{i=t+1}^{\ell} |A(L_i)| \leq \sigma(\mathcal{H}) + \sigma(\mathcal{M})$, then by Lemma 2,

$$\begin{aligned} |C| &\geq \frac{1}{2}(\sigma(\mathcal{H}) + \sigma(\mathcal{M}) + (b-1) \sum_{i=t+1}^{\ell} |A(L_i)|)^r + 4 \\ &\geq \frac{1}{2}(4d\sigma(\mathcal{M}) + 1)^r + 3 \\ &\geq \frac{1}{2}n^r + 3. \end{aligned}$$

So assume $\sum_{i=t+1}^{\ell} |A(L_i)| \geq \sigma(\mathcal{H}) + \sigma(\mathcal{M})$. Applying Lemma 2 again, we have

$$\begin{aligned} |C| &\geq \frac{1}{2}(\sum_{i=t+1}^{\ell} |A(L_i)| + (b-1)(\sigma(\mathcal{H}) + \sigma(\mathcal{M})))^r + 3 \\ &\geq \frac{1}{2}(4d\sigma(\mathcal{M}) + 1)^r + 3 \\ &\geq \frac{1}{2}n^r + 3. \end{aligned}$$

Case 2. $\mathcal{M}$ may be chosen so that $x \notin \{x_0, y_0\} \cap \{c_0, d_0\}$, $\{x_0, y_0\} \neq \{c_0, d_0\}$, and $x_0 y_0 \in \mathcal{H}$.

Assume that $c_0 d_0 \in E(H_s) - E(H_{s-1})$ and $x_0 y_0 \in E(H_t) - E(H_{t-1})$. We only consider the case $s \leq t$; since the case $t \geq s$ is similar.

We claim that there is a path P_0 in $\mathcal{H}$ from x to some $z \in \{c_0, d_0\} \cup \{x_0, y_0\}$ such that

(a) $|E(P_0)| \geq \frac{1}{2}(\frac{|H_0|+1}{d})^r$,
(b) $c_0d_0 \in E(P_0)$ or $x_0y_0 \in E(P_0)$, and
(c) if $c_0d_0 \in E(P_0)$ then $z \in \{x_0, y_0\}$, and $x_0y_0 \notin E(P_0)$ unless $\{x_0, y_0\} = \{c_0, d_0\}$, and if $x_0y_0 \in E(P_0)$ then $z \in \{c_0, d_0\}$, and $c_0d_0 \notin E(P_0)$ unless $\{x_0, y_0\} = \{c_0, d_0\}$.

If $s = 0$, then this claim follows from (i) and (ii) of Corollary 1, with c_0d_0, x_0y_0 as uv, pq, respectively. So assume $s \geq 1$. In $\bigcup_{i=0}^{s-1} H_i$, we use (c) of Theorem 2 to find a path Q from a_{s-1} to b_{s-1} such that $|E(Q)| \geq \frac{1}{2}|H_0|^r + 2 > \frac{1}{2}(\frac{|H_0|+1}{d})^r$. By applying the same argument as for (1) in Case 2 in the proof of Lemma 10, we find a path R from x to $z \in \{c_0, d_0\} \cup \{x_0, y_0\}$ such that $a_{s-1}b_{s-1} \in E(R)$ and (b) and (c) hold. Now $P_0 := (Q - a_{s-1}b_{s-1}) \cup R$ gives the desired path.

Suppose $x_0y_0 \in E(P_0)$ and, without loss of generality, assume $z = c_0$. Let $y' \in N(y)$ which is contained in the extreme 3-block in L_ℓ. By Lemma 8 there is a path P_1 in $\mathcal{L} - d_0$ from c_0 to y' such that $|E(P_1)| \geq \frac{1}{2}(\frac{\sigma(\mathcal{L})}{d})^r + 1$. By Lemma 7, there is a path P_2 from x_0 to y_0 in $\mathcal{M}$ such that $|E(P_2)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{M})}{d})^r + 2$. Let C be the cycle obtained from $(P_0 \cup P_1) + \{y, yy', yx\}$ by replacing x_0y_0 with P_2. Then

$$\begin{aligned}
|C| &\geq |E(P_0)| + |E(P_1)| + |E(P_2)| + 1 \\
&\geq \frac{1}{2}[(\frac{|H_0|}{d} + \sigma(\mathcal{L}))^r + (\frac{(d-1)\sigma(\mathcal{M})}{d})^r] + 4 \\
&\geq \frac{1}{2}[\frac{|H_0|}{d} + \sigma(\mathcal{L}) + ((b-1)(d-1)/d)\sigma(\mathcal{M})]^r + 4 \\
&\geq \frac{1}{2}[4(d-1)\sigma(\mathcal{M}) + 1]^r + 3 \\
&\geq \frac{1}{2}n^r + 3.
\end{aligned}$$

Now assume $c_0d_0 \in E(P_0)$ and, without loss of generality, assume $z = x_0$. Let y' be a neighbor of y which belongs to the extreme 3-block contained in M_m. By Lemma 8, there is a path P_1 from x_0 to y' in $\mathcal{M} - y_0$ such that $|E(P_1)| \geq \frac{1}{2}(\frac{\sigma(\mathcal{M})}{d})^r + 1$. By Lemma 7, there is a path P_2 from c_0 to d_0 in $\mathcal{L}$ such that $|E(P_2)| \geq \frac{1}{2}(\frac{d-1}{d}\sigma(\mathcal{L}))^r + 2$. Let C be the cycle obtained from $(P_0 \cup P_1) + \{y, yy', yx\}$ by replacing c_0d_0 with P_2. Then

$$\begin{aligned}
|C| &\geq |E(P_0)| + |E(P_1)| + |E(P_2)| + 1 \\
&\geq \frac{1}{2}[(\frac{|H_0|}{d} + \sigma(\mathcal{M}))^r + (\frac{d-1}{d}\sigma(\mathcal{L}))^r] + 4.
\end{aligned}$$

If $\frac{d-1}{d}\sigma(\mathcal{L}) \geq \frac{|H_0|}{d} + \sigma(\mathcal{M})$, we have by Lemma 2 that

$$|C| \geq \frac{1}{2}[(b-1)(\frac{|H_0|}{d} + \sigma(\mathcal{M})) + \frac{d-1}{d}\sigma(\mathcal{L}))^r + 4 \geq \frac{1}{2}[4(d-1)\sigma(\mathcal{M}) + 1]^r + 3 \geq \frac{1}{2}n^r + 3.$$

If $\frac{d-1}{d}\sigma(\mathcal{L}) < \frac{|H_0|}{d} + \sigma(\mathcal{M})$, then by Lemma 2 and because $\sigma(\mathcal{L}) \geq \sigma(\mathcal{M})$ we have

$$|C| \geq \frac{1}{2}[\frac{|H_0|}{d} + \sigma(\mathcal{M}) + (b-1)\frac{d-1}{d}\sigma(\mathcal{L})]^r + 4 \geq \frac{1}{2}[4(d-1)\sigma(\mathcal{M}) + 1]^r + 3 \geq \frac{1}{2}n^r + 3.$$

Case 3. $\mathcal{M}$ may be chosen so that $\{x_0, y_0\} = \{c_0, d_0\}$, $x \notin \{c_0, d_0\}$, and $\{c_0, d_0\}$ is not a cut of $\mathcal{H}$.

We first show that there is a path Q_0 in $\mathcal{H}$ from x to $\{c_0, d_0\}$ such that $\{c_0, d_0\} \not\subseteq V(Q_0)$ and $|E(Q_0)| \geq \frac{1}{2}(\frac{|H_0|}{d})^r + 1$. If $\{c_0, d_0\} \not\subseteq V(H_0)$, then since $\{c_0, d_0\}$ is not a cut in $\mathcal{H}$, we get Q_0 by finding a path from x to $\{c_0, d_0\}$ through a_0b_0 and finding a path in H_0 from a_0 to b_0 of length at least $\frac{1}{2}|H_0|^r + 1$ (by (c) of Theorem 2). So assume $\{c_0, d_0\} \subseteq V(H_0)$. In $H_0 + a_0c_0$, we apply (b) of Theorem 2 to find cycle C_0 through a_0c_0 and c_0d_0 of length at least $\frac{1}{2}(\frac{|H_0|}{d})^r + 3$. It is then easy to see that $C_0 - d_0$ can be extended to the desired path Q_0.

By Lemma 8, there is a path Q_1 in $\mathcal{M} - d_0$ from c_0 to some $y' \in N(y) \cap V(M_m)$ such that $|E(Q_1)| \geq \frac{1}{2}(\frac{\sigma(\mathcal{M})}{d})^r + 1$. By Lemma 7, there is a path Q_2 from c_0 to d_0 in $\mathcal{L}$ such that $|E(Q_2)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{L})}{d})^r + 2$.

Let $C := (Q_0 \cup Q_2 \cup Q_1) + \{y, yx, yy'\}$. Then by Lemma 2, we have

$$|C| \geq |E(Q_0)| + |E(Q_1)| + |E(Q_2)| + 1 \geq \frac{1}{2}[(\frac{|H_0|}{d} + \sigma(\mathcal{M}))^r + (\frac{d-1}{d}\sigma(\mathcal{L}))^r] + 5.$$

By the same argument as in Case 2, we have $|C| \geq \frac{1}{2}n^r + 3$.

Case 4. For every choice of $\mathcal{M}$ such that $\sigma(\mathcal{M}) \geq \frac{n}{4(d-2)}$, we have $x \in \{c_0, d_0\} \cap \{x_0, y_0\}$ or $\{x_0, y_0\} = \{c_0, d_0\}$ is a cut in $\mathcal{H}$.

In this case, we see that the sum of $\sigma(\mathcal{M})$ for those $\mathcal{HL}$-legs $\mathcal{M}$ in previous cases is at most $\frac{n-1}{2}$. Hence the sum of $\sigma(\mathcal{M})$ for those $\mathcal{HL}$-legs $\mathcal{M}$ for which $x \in \{c_0, d_0\} \cap \{x_0, y_0\}$ or $\{x_0, y_0\} = \{c_0, d_0\}$ is a cut in $\mathcal{H}$ is at least $\frac{n}{4}$. Let k denote the number of $\mathcal{HL}$-legs $\mathcal{M}$ for which $x \in \{c_0, d_0\} \cap \{x_0, y_0\}$ or $\{x_0, y_0\} = \{c_0, d_0\}$ is a cut in $\mathcal{H}$. Let $z = x$ if $x \in \{c_0, d_0\} \cap \{x_0, y_0\}$, and otherwise let $z \in \{c_0, d_0\}$. Without loss of generality, we assume that $z = x_0 = c_0$.

Let $t(\mathcal{M}) := d_{\mathcal{M}}(z) - 1$. By induction on k, we can show that $\max\{\frac{\sigma(\mathcal{M})}{t(\mathcal{M})}\} \geq \frac{n}{4k}$. Since $k \leq d-1$, we have $(d-1)\max\{\frac{\sigma(\mathcal{M})}{t(\mathcal{M})}\} \geq \frac{n}{4}$. So we further choose $\mathcal{M}$ so that $\frac{\sigma(\mathcal{M})}{t(\mathcal{M})}$ is maximum, and so, $\frac{\sigma(\mathcal{M})}{t(\mathcal{M})} \geq \frac{n}{4k}$.

Let G^* denote the graph obtained from G by deleting those components of $G - \{x_0, y_0, y\}$ which contain no vertex of $\mathcal{M}$. Then $|G^*| \geq \sigma(\mathcal{M}) + 1$. Note that $G^* + \{yx_0, yy_0, x_0y_0\}$ is 3-connected and has maximum degree at most $d+1$, and any vertex of degree $d+1$ must be incident with x_0y or x_0y_0.

Suppose $y_0 \in V(\mathcal{H})$. Let Q_0 be a path from x to y_0 in $\mathcal{H}$ through edge c_0d_0. By Corollary 7, there is a path Q_1 from $x = c_0$ to d_0 in $\mathcal{L}$ such that $|E(Q_1)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{L})}{d})^r + 2$. By (a) of Theorem 2, there is a path Q_2 from y_0 to y in $G^* - x_0$ such that $|E(Q_2)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{M})}{dt(\mathcal{M})})^r + 1$. Let $C := (Q_0 \cup Q_1 \cup Q_2) + yx$. Then

$$|C| = |E(Q_1)| + |E(Q_2)| + 2 \geq \frac{1}{2}[(\frac{(d-1)\sigma(\mathcal{L})}{d})^r + (\frac{(d-1)\sigma(\mathcal{M})}{dt(\mathcal{M})})^r] + 5.$$

Since $\sigma(\mathcal{L}) \geq \sigma(\mathcal{M})$ and by Lemma 2, we have

$$|C| \geq \frac{1}{2}[\frac{(d-1)\sigma(\mathcal{L})}{d} + \frac{(b-1)(d-1)\sigma(\mathcal{M})}{dt(\mathcal{M})}]^r + 4$$

$$\geq \frac{1}{2}[4(d-1)\sigma(\mathcal{M})/t(\mathcal{M})]^r + 4$$
$$\geq \frac{1}{2}n^r + 3.$$

Thus, we may assume $y_0 \notin V(\mathcal{H})$. Then $y_0 \in V(L_1)$. Let $n^* \sum_{i=2}^{\ell} |A(L_i)|$. By our choice of $\mathcal{L}$, we have $n^* \geq \sigma(\mathcal{M})$. Let Q_0 be a path from x to y_0 through c_1d_1 in L_1. Since L_1 is 2-connected, Q_0 exists. By Corollary 7 there is a path Q_1 from c_1 to d_1 in $\bigcup_{i=2}^{\ell} L_i$ such that $|E(Q_1)| \geq \frac{1}{2}(\frac{(d-1)n^*}{d})^r + 1$. Define G^* as above. By (a) of Theorem 2 there is a path Q_2 in $G^* - x_0$ from y_0 to y such that $|E(Q_2)| \geq \frac{1}{2}(\frac{(d-1)\sigma(\mathcal{M})}{td(\mathcal{M})})^r + 1$. Let C be the cycle obtained from $(Q_0 \cup Q_2) + yx$ by replacing the edge c_1d_1 by Q_1. By the same argument as in the above paragraph (with $\sigma(\mathcal{L})$ replaced by n^*), we can show that $|C| \geq \frac{1}{2}n^r + 3$. □

6 Conclusions

We now complete the proof of Theorem 2. Let n, d, r, G be given as in Theorem 2. We apply induction on n. When $n = 5$, G is isomorphic to one the following three graphs: K_5, K_5 minus an edge, or the wheel on five vertices. In each case, we can verify that Theorem 2 holds. So assume that $n \geq 6$ and Theorem 2 holds for all 3-connected graphs with at most $n - 1$ vertices. Then (a) of Theorem 2 holds by Lemma 9, (b) of Theorem 2 holds by Lemma 10, and (c) of Theorem 2 holds by Lemma 11. This completes the proof of Theorem 2. □

Our proof of Theorem 2 implies a polynomial time algorithm which, given a 3-connected n-vertex graph, finds a cycle of length $\frac{1}{2}n^r + 3$. In fact, our proof implies a cubic algorithm when combined with following two results from [7].

Lemma 12. *Let G be a k-connected graph, where k is a positive integer. Then G contains a k-connected spanning subgraph with $O(|G|)$ edges, and such a subgraph can be found in $O(|G|)$ time.*

The next result is an easy consequence of a result in [7], which states that, in a 2-connected graph G, one can find, in $O(|G|)$ time, two disjoint paths between two given vertices.

Lemma 13. *Let G be a 2-connected graph and let $e, f \in E(G)$. Then there is a cycle through e and f in G, and such a cycle can be found in $O(|G|)$ time.*

Finally, we give an outline of the desired algorithm. Let G be a 3-connected graph with maximum degree at most d, let $e = xy \in E(G)$, and assume $|G| \geq 5$. The following procedure finds a cycle C through e in G with $|C| \geq \frac{1}{2}|G|^r + 3$.

1. **Preprocessing**: Replace G with a 3-connected spanning subgraph of G with $O(|G|)$ edges.
2. We either find the desired cycle C, or we reduce the problem to (a), (b) or (c) of Theorem 2 for some 3-connected graphs G_i, for which $|G_i| < |G|$ and each G_i contains a vertex which does not belong to any other G_i.

3. Replace each G_i with a 3-connected spanning subgraph of G_i with $O(|G_i|)$ edges.
4. Apply Lemma 9 to those G_i for which (a) of Theorem 2 needs to be applied. Apply Lemma 10 to those G_i for which (b) of Theorem 2 needs to be applied. Apply Lemma 11 to those G_i for which (c) of Theorem 2 needs to be applied.
5. Repeat step 3 and step 4 for new 3-connected graphs.
6. In the final output, replace all virtual edges by paths in G to complete the desired cycle C.

It can be shown that the algorithm runs in $O(|G|^3)$ time.

References

1. Björklund, A., Husfeldt, T.: Finding a path of superlogarithmic length. SIAM J. Comput. **32** (2003) 1395-1402.
2. Chen,G., Xu, J., Yu, X.: Circumference of graphs with bounded degree. SIAM J. Comput. **33** (2004) 1136-1170.
3. Feder, T., Motwani, R., and Subi, C.: Approximating the longest cycle problem in sparse graphs. SIAM J. Comput. **31** (2002) 1596-1607.
4. Feder, T., Motwani, R.: Finding a long cycle in a graph with a degree bound and a 3-cyclable minor. Manuscript (2004).
5. Gabow, H.N.: Finding paths and cycles of superpolylogarithmic length. Proc. 36th annual ACM symposium on Theory of Computing (STOC), Chicago, 2004, pp. 407-416.
6. Hopcroft, J.E., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM J. Comput. **2** (1973) 135-158.
7. Ibaraki, T., Nagamochi, H.: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica **7** (1992) 583-596.
8. Jackson, B., Wormald, N.C.: Longest cycles in 3-connected graphs of bounded maximum degree. in: Graphs, Matrices, and Designs (R. S. Rees, ed.), Marcel and Dekker, Inc (1993) 237-254.
9. Karger, D., Motwani, R., Ramkumar, G.D.S.: On approximating the longest path in a graph. Algorithmica **18** (1997) 82-98.
10. Tutte, W.T.: Connectivity in Graphs. University of Toronto Press (1966).
11. Whitney, H.: A theorem on graphs. Ann. of Math. **32** (1931) 378-390.

Bin Packing and Covering Problems with Rejection

Yong He[1,⋆] and György Dósa[2]

[1] Department of Mathematics, and State Key Lab of CAD & CG, Zhejiang University, Hangzhou 310027, P.R. China
mathhey@zju.edu.cn
[2] Department of Mathematics, University of Veszprém, Hungary
dosagy@almos.vein.hu

Abstract. In this paper we consider the following problems: We are given a set of n items $\{u_1, \cdots, u_n\}$, each item u_i is characterized by its size $w_i \in (0, 1]$ and its penalty/profit $p_i \geq 0$, and a number of unit-capacity bins. An item can be either rejected, in which case we pay/get its penalty/profit, or put into one bin under the constraint that the total size of the items in the bin is not greater/smaller than 1. No item can be spread into more than one bin. The objective is to minimize/maximize the sum of the number of used/covered bins and the penalties/profits of all rejected items. We call the problems bin packing/covering with rejection penalties/profits, and denoted by BPR and BCR respectively. For the online BPR problem, we present an algorithm with an absolute competitive ratio of 2.618 while the lower bound is 2.343, and an algorithm with an asymptotic competitive ratio of arbitrarily close to 7/4 while the lower bound is 1.540. For the offline BPR problem, we present an algorithm with an absolute worst-case ratio of 2 while the lower bound is 3/2, and an algorithm with an asymptotic worst-case ratio of 3/2. For the online BCR problem, we show that no algorithm can have an absolute competitive ratio of greater than 0, and present an algorithm with an asymptotic competitive ratio of 1/2, which is the best possible. For the offline BCR problem, we also present an algorithm with an absolute worst-case ratio of 1/2 which matches the lower bound.

1 Introduction

In this paper we consider a variant of the classical bin packing problem which has the special feature that items can be rejected at a certain cost. We are given a set of n items $\{u_1, \cdots, u_n\}$, each item u_i is characterized by its size $w_i \in (0, 1]$ and its penalty $p_i \geq 0$, and a number of unit-capacity bins. The cost of purchasing one bin is 1. An item can be either rejected, in which case we pay its penalty, or put into one bin under the constraint that the total size of the items in the bin, called the *content* of the bin, is not greater than 1. No item can be spread into more than one bin. The objective is to minimize the sum of the cost for

⋆ Supported by the NSFC (10271110, 60021201) and TRAPOYT of China

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 885–894, 2005.

purchasing bins and the penalties of all rejected items. We call this problem *bin packing with rejection penalties* or BPR for short.

This problem may have applications in the real world. Let us first consider a company's intranet. Usually it has two main functions. One is to release the company's messages, providing services for users. The other one is to help employees of this company use it to visit outside internet for getting useful files and messages [6]. If a message or file on outside internet is visited by employees many times, the administrator of the intranet may download it to a web server of the intranet to decrease communication cost and traffic. Note that the capacity of any web server is limited. If the content in one web server is close to its limit, a new web server must be purchased. This yields the following problem: we have a list of messages and files which are considered for downloading, each with a size and penalty, here penalty denotes its estimated communication cost if it is not downloaded. The goal is to minimize the total cost, i.e., the sum of the cost for purchasing web servers and the total penalties for not downloaded messages and files. Clearly this problem can be reduced to BPR. Take another example, a transportation company manages to transmit goods. Two choices are presented. The one is to pack goods in its own trucks, with a constraint that the total size of the packed goods in a truck is not greater than truck's capacity. The other one is to ask a forwarding agent to deliver the goods. For the first choice, the cost is proportional to the number of used trucks, while for the second choice, the company should pay the forwarding agent costs depending on goods. Assume that all own trucks have the same capacity and the same cost 1. Thus the total cost consists of one for used trucks and one for all forwarded goods. The goal is to transport all goods in the cheapest way, i.e., deciding which goods are packed in its own trucks or forwarded to minimize the total cost. It is clear that this problem is essentially the bin packing problem with rejection penalties.

We also consider the dual problem of BPR in this paper, which can be described as follows: We are given a set of n items $\{u_1, \cdots, u_n\}$, each item u_i is characterized by its size $w_i \in (0, 1]$ and its profit $p_i \in [0, 1)$, and a number of unit-capacity bins. An item can be either rejected, in which case we get its profit, or assigned to one bin. If one bin has a content of at least 1, then we say that it is *covered* and get profit 1. No item can be spread into different bins. The objective is to maximize the total profit, i.e., the sum of the number of covered bins and the profits of all rejected items. This problem can also be viewed as a variant of the classical bin covering problem, hence we also refer it to *bin covering with rejection profits* or BCR for short. The problem BCR may model the following application. An industry company holds several monopolies. It can choose to break them up into smaller companies, each of which must be large enough to be viable, or sell them to get profit. Suppose that the profit of a viable company is the same. The objective is to maximize the total profit, i.e., the number of viable companies and the profit of those monopolies which are sold.

For the problem BPR (BCR), if all $p_i > 1$ ($p_i = 0$), $i = 1, \cdots, n$, then no item can be rejected in an optimal solution. Hence the bin packing (bin covering) problem is a special case of BPR (BCR). It follows that the problems under

consideration are strongly NP-hard [7]. The bin packing problem is a classical combinatorial optimization problem that has been extensively studied for more than three decades. The readers may refer to survey papers [2],[5] and papers cited therein. Meanwhile, the bin covering problem is also well-studied since it was first proposed by Assman et al. [1]. The readers may refer to recent papers [3],[8], survey paper [5] and papers cited therein. But to our best knowledge, both BPR and BCR are unexplored.

In the *online* version of the discussed problems, item arrives one by one, and the decision to either reject an item or pack it into one bin has to be made before any information about the next item is revealed. If we are allowed to make decisions with full information of the set of items, the version is called *offline*. Algorithms for online/offline problem are called *online/offline algorithms*.

The quality of an offline approximation algorithm is usually measured by its worst-case ratio, while online algorithm by competitive ratio. Let $A(I)$ denote the objective function value produced by an algorithm A, and $OPT(I)$ denote the optimal value in the offline version. Then for BPR, the *absolute worst-case* (*competitive*) *ratio* of A is defined by $R_A = \sup_I\{\frac{A(I)}{OPT(I)}\}$; and the *asymptotic worst-case* (*competitive*) *ratio* of A is defined by $R_A^\infty = \limsup_{n\to\infty} \max\{\frac{A(I)}{OPT(I)} \mid OPT(I) = n\}$. For BCR, the absolute worst-case (competitive) ratio and asymptotic worst-case (competitive) ratio of A, are respectively defined by $R_A = \inf_I\{\frac{A(I)}{OPT(I)}\}$, and $R_A^\infty = \liminf_{n\to\infty} \min\{\frac{A(I)}{OPT(I)} \mid OPT(I) = n\}$. An offline (online) minimization/maximization problem has a *lower/upper bound* ρ with respect to absolute or asymptotic worst-case (competitive) ratio if no offline (online) algorithm has an absolute or asymptotic worst-case (competitive) ratio of smaller/greater than ρ, respectively. An offline (online) algorithm is called *best possible* if its worst-case (competitive) ratio matches the corresponding lower/upper bound of the minimization/maximization problem.

Table 1. Summary of the results for BPR and BCR

	BPR		BCR	
	upper bound	lower bound	lower bound	upper bound
online, absolute ratio	2.618	2.343	0	0
online, asymptotic ratio	7/4	1.540 [13]	1/2	1/2 [4]
offline, absolute ratio	2	3/2 [7]	1/2	1/2 [1]
offline, asymptotic ratio	3/2	open	1/2	open

In this paper we study the problems BPR and BCR. Both online and offline versions are considered. The results are listed in Table 1. Algorithms $RFF1 - RFF4$ and $MDNFD$ run in time $O(n \log n)$, and algorithm $MDNF$ runs in time $O(n)$. For comparison purposes, we also list the known best results of bin packing and covering problems in Table 2. As we know, even for the classical bin packing and covering problems, it took a long time to get most of the results in

Table 2. Summary of the known best results for bin packing and covering problems

	Bin packing problem		Bin covering problem	
	upper bound	lower bound	lower bound	upper bound
online, absolute ratio	7/4 [15]	5/3 [14]	1/2 [1]	open
online, asymptotic ratio	1.589 [11]	1.540 [13]	1/2 [1]	1/2 [4]
offline, absolute ratio	3/2 [12]	3/2 [7]	1/2 [1]	1/2 [1]
offline, asymptotic ratio	*FPTAS* [10]	1	*FPTAS* [8]	1

Table 2, and it is still open how to close the existing gaps. Our problems under consideration become more complicated, and harder to approximate since one more parameter is introduced for every item. To devise the algorithms presented in this paper, we will introduce several strategies for trade-off between packing cost/profit and rejection penalty/profit. Furthermore we will employ harmonic technique with consideration of penalty parameter. In analysis of the algorithms, we will develop methods to estimate the optimal value. Especially we will apply a simple linear programming technique instead of case by case analysis to prove the asymptotic competitive ratio of algorithm $RFF2$ (see Theorem 4).

In the remainder of this paper, denote $W(S) = \sum_{u_i \in S} w_i$ and $P(S) = \sum_{u_i \in S} p_i$ for an item set S. Denote $M_1 = \{u_i | p_i/w_i > 1\}$ and $M_2 = \{u_i | p_i/w_i \leq 1\}$. Denote by $\epsilon > 0$ a sufficiently small number, and by N a sufficiently large positive integer, whose exact values are immaterial in later proofs. In the online version of the problems, before all items arriving, we do not know which of them are in M_1 and M_2, but we still use the notation $u_i \in M_1$ ($u_i \in M_2$) to mean that the item u_i satisfies $p_i/w_i > 1$ ($p_i/w_i \leq 1$) for simplicity.

2 The Problem BPR

2.1 Preliminaries

We use I' to denote an instance of the bin packing problem, and $FF(I')$ to denote the number of bins used by *First Fit* (FF) algorithm.

Lemma 1. *Let I' be an instance of the bin packing with $|I'| = n'$.*
(1) *if $w_i \leq 1/2, i = 1, \cdots, n'$, then*

$$FF(I') \begin{cases} = 1, & if \quad \sum_{i=1}^{n'} w_i \leq 1, \\ < \frac{3}{2} \sum_{i=1}^{n'} w_i + \frac{1}{2}, & if \quad \sum_{i=1}^{n'} w_i > 1. \end{cases}$$

$$\text{(2)([2])} \qquad \text{FF(I')} \begin{cases} = 1, & if \quad \sum_{i=1}^{n'} w_i \leq \frac{1}{2}, \\ < 2 \sum_{i=1}^{n'} w_i, & if \quad \sum_{i=1}^{n'} w_i > \frac{1}{2}. \end{cases}$$

Theorem 1. *For BPR, $OPT(I) \geq W(M_1) + P(M_2)$.*

2.2 Online Algorithms

Algorithm $RFF1$:

1. $k = 1, P = 0$. Denote $\phi = (\sqrt{5} - 1)/2 \approx 0.618$.
2. If no new item arrives, stop. Else go to 3.
3. If $P + p_k < \phi$, then reject u_k, set $P = P + p_k$, $k = k + 1$ and go to 2; Otherwise, go to 4.
4. If $u_k \in M_1$, then pack it by FF algorithm; Otherwise, reject it. Set $k = k+1$. If no new item arrives, stop. Else go back to 4.

Theorem 2. $R_{RFF1} = (1 + \phi)/\phi = 2 + \phi \approx 2.618$.

Next we consider the lower bound of the online problem BPR in terms of the absolute competitive ratio. Let $\beta \approx 0.7446, x \approx 0.2991$ be the root of the following system of equations: $\frac{2+\beta+x}{1+x} = \frac{1+\beta}{\beta} = \beta + 2x + 1$.

Theorem 3. *No online algorithm can have an absolute competitive ratio of less than* $(1 + \beta)/\beta \approx 2.343$.

Now we turn to study the asymptotic competitive ratios of online algorithms. To show the asymptotic competitive ratio of $RFF1$, we consider the following sequence with $2N+1$ items: $w_1 = \epsilon, p_1 = \phi - \epsilon$, $w_2 = \cdots = w_{2N+1} = \epsilon + 1/2, p_2 = \cdots = p_{2N+1} = 2\epsilon + 1/2$. Then $RFF1$ only rejects the first item while an optimal solution only accepts the first two items. Therefore $\frac{RFF1(I)}{OPT(I)} = \frac{2N+\phi-\epsilon}{N+4N\epsilon-2\epsilon+1/2} \to 2$ $(N\epsilon \to 0, N \to \infty)$. It follows that the asymptotic competitive ratio of $RFF1$ cannot be smaller than 2. In the following, we present another online algorithm $RFF2$ with an asymptotic competitive ratio of arbitrarily close to 7/4.

Let m be a large positive integer. We partition set M_1 into several subsets as follows:

$$M_{11} = \{u_i \in M_1 | w_i \le \tfrac{1}{2}\},$$
$$M_{jk} = \left\{u_i \in M_1 | \tfrac{m}{j} < w_i \le \tfrac{m}{j-1}, \tfrac{k-1}{m} < \tfrac{p_i}{w_i} \le \tfrac{k}{m}\right\},$$
$$j = m+1, \cdots, 2m, k = m+1, \cdots, j-1,$$
$$M_{jj} = \{u_i \in M_1 | \tfrac{m}{j} < w_i \le \tfrac{m}{j-1}, \tfrac{j-1}{m} < \tfrac{p_i}{w_i}\}, j = m+1, \cdots, 2m.$$

Algorithm $RFF2$:

1. If the incoming item is in M_{11}, pack it by FF algorithm.
2. If the incoming item is in one of the sets M_{jk}, $j = m+1, \cdots, 2m$, $k = m+1, \cdots, j-1$, or M_2, reject it.
3. If the incoming item is in $M_{jj}, m+1 \le j \le 2m$, then pack it each into a bin, and this bin will not be used to pack any other item.

Theorem 4. *For any given positive integer* $m \ge 2$, $R^{\infty}_{RFF2} \le \frac{7m-3}{4m-2}$, *hence there exists an online algorithm with an asymptotic competitive ratio of arbitrarily close to* 7/4.

Proof. Since all items in M_{11} have sizes no greater than 1/2, and are packed by FF algorithm, we have

$$RFF2(M_{11}) = FF(M_{11}) \le \max\left\{\frac{3}{2}W(M_{11}) + \frac{1}{2}, 1\right\} \le \frac{3}{2}W(M_{11}) + 1$$

by Lemma 1(1). By the definition of M_{jk}, $j = m+1, \cdots, 2m$, $k = m+1, \cdots, j-1$, we have $P(M_{jk}) \le \frac{k}{m}W(M_{jk})$. Because every item in M_{jj} does not share a bin with any other item in $RFF2$ algorithm, and for any item $u_i \in M_{jj}$, $m/j < w_i$ holds, i.e. $jw_i/m > 1$, we obtain that the number of bins used for the items in M_{jj} is $|M_{jj}| \le \sum_{u_i \in M_{jj}} \frac{j}{m} w_i = \frac{j}{m}W(M_{jj})$. Therefore we have

$$RFF2(I) \le \frac{3}{2}W(M_{11}) + 1 + \sum_{m+1\le j\le 2m, m+1\le k\le j-1} \frac{k}{m}W(M_{jk}) + \sum_{m+1\le j\le 2m} \frac{j}{m}W(M_{jj}) + P(M_2). \quad (1)$$

By Theorem 1, we have

$$OPT(I) \ge W(M_{11}) + \sum_{m+1\le j\le 2m, m+1\le k\le j-1} W(M_{jk}) + \sum_{m+1\le j\le 2m} W(M_{jj}) + P(M_2). \quad (2)$$

Next we are going to obtain another lower bound of $OPT(I)$. Let

$$I' = I \setminus (M_{11} \cup M_2) = \bigcup_{j=m+1,\cdots,2m, k=m+1,\cdots,j} M_{jk}.$$

It is obvious that $OPT(I') \le OPT(I)$. Let us consider an optimal solution for instance I'. For every item $u_i \in M_{jk} \subseteq I'$, $j = m+1, \cdots, 2m$, $k = m+1, \cdots, j-1$, since $w_i > 1/2$, this item cannot share a bin with any other item in I', if it is accepted in the optimal solution. In this case, its contribution to the optimal value is $1 \ge \frac{j-1}{m}w_i$. If this item is rejected in the optimal solution, the contribution to the optimal value is $p_i \ge \frac{k-1}{m}w_i$. Hence we conclude that the contribution of item u_i to the optimal value is at least $\min\left\{\frac{j-1}{m}, \frac{k-1}{m}\right\} w_i = \frac{k-1}{m}w_i$. Similarly, for any $u_i \in M_{jj}, j = m+1, \cdots, 2m$, its contribution to the optimal value of I' is at least $\frac{j-1}{m}w_i$. Therefore we obtain

$$OPT(I) \ge \sum_{m+1\le j\le 2m, m+1\le k\le j-1} \frac{k-1}{m}W(M_{jk}) + \sum_{m+1\le j\le 2m} \frac{j-1}{m}W(M_{jj}). \quad (3)$$

To get the asymptotic competitive ratio, we need to show that there exist constants a, b such that $RFF2(I)/OPT(I) \le a + b/OPT(I)$ for any instance I.

We are interested in the maximum possible value of $RFF2(I)/OPT(I)$. Dividing the inequalities (1)-(3) by $OPT(I)$, we construct a linear program as follows

$$\begin{aligned} \max \quad & z = \tfrac{3}{2}x_{11} + \sum_{m+1\le j\le 2m, m+1\le k\le j-1} \tfrac{k}{m}x_{jk} + \sum_{m+1\le j\le 2m} \tfrac{j}{m}x_{jj} + y \\ s.t. \quad & x_{11} + \sum_{m+1\le j\le 2m, m+1\le k\le j-1} x_{jk} + \sum_{m+1\le j\le 2m} x_{jj} + y \le 1, \\ & \sum_{m+1\le j\le 2m, m+1\le k\le j-1} \tfrac{k-1}{m}x_{jk} + \sum_{m+1\le j\le 2m} \tfrac{j-1}{m}x_{jj} \le 1, \\ & y \ge 0, x_{11} \ge 0, x_{jk} \ge 0, j = m+1, \cdots, 2m, k = m+1, \cdots, j, \end{aligned} \tag{4}$$

where $x_{11} = \frac{W(M_{11})}{OPT(I)}, x_{jk} = \frac{W(M_{jk})}{OPT(I)}$, $j = m+1, \cdots, 2m, k = m+1, \cdots, j, y = \frac{P(M_2)}{OPT(I)}$ and $z = \frac{RFF2(I)}{OPT(I)}$ are variables. Note that we omit the additive factor $1/OPT(I)$ from the objective function of the linear program, since we are considering the asymptotic competitive ratio. It is clear that the asymptotic competitive ratio is not greater than the optimal value of (4).

By a simple calculation, we know that (4) has a unique optimal solution: $x_{11} = \frac{m-1}{2m-1}, x_{2m,2m} = \frac{m}{2m-1}$, and the values of all remaining variables equal to 0. The optimal value is $\frac{3}{2}\frac{m-1}{2m-1} + 2\frac{m}{2m-1} = \frac{7m-3}{4m-2}$ which can be arbitrarily close to $7/4$ if m is chosen to be large enough. We thus finish the proof. □

Since no online algorithm for the classical bin packing problem can have an asymptotic competitive ratio of less than 1.540 [13], it is still true for BPR.

2.3 Offline Algorithms

Algorithm $RFF3$:

1. For $\sum_{i=1}^{n} w_i \le 1$, if $\sum_{i=1}^{n} p_i > 1$, then all items are accepted and packed into one bin, otherwise reject all jobs.
2. For $\sum_{i=1}^{n} w_i > 1$, if $u_i \in M_1, i = 1, 2, \cdots, n$, then pack it by FF algorithm, otherwise reject it.

Theorem 5. $R_{RFF3} = R^{\infty}_{RFF3} = 2$.

Note that there does not exist a polynomial time algorithm with an absolute worst-case ratio of smaller than $3/2$ for the classical bin packing problem, unless P=NP [7], it still holds for the problem BPR. In the remainder of this section, we are devoted to presenting an offline algorithm with an asymptotic worst-case ratio of $3/2$.

First we split M_1 into four subsets as follows:
$M_{1h} = \left\{u_i \in M_1 : w_i > \frac{2}{3}\right\}$, $M_{1l} = \left\{u_i \in M_1 : \frac{1}{2} < w_i \le \frac{2}{3}\right\}$,
$M_{1m} = \left\{u_i \in M_1 : \frac{1}{3} < w_i \le \frac{1}{2}\right\}$, $M_{1s} = \left\{u_i \in M_1 : w_i \le \frac{1}{3}\right\}$,
and the items in $M_{1h}, M_{1l}, M_{1m}, M_{1s}$ are called as *huge, large, medium* and *small* items respectively. We propose a procedure for pre-process as below, which puts one large item and one medium item pairwise into one bin as much as possible. Denote by $\max(I)$ the maximum number k such that k large items and k medium items can be packed pairwise into k bins for instance I.

Lemma 2. *Let $u_{l\alpha}$ be the largest large item for which there exists a medium item u_{mj} such that $w_{l\alpha} + w_{mj} \leq 1$, and let $u_{m\beta}$ be an arbitrary medium item satisfying $w_{l\alpha} + w_{m\beta} \leq 1$. Then* $\max(I \setminus \{u_{l\alpha}, u_{m\beta}\}) = \max(I) - 1$.

By Lemma 2 we can describe the pre-process procedure, which puts large and medium items commonly into bins as many as possible in a greedy way.

Procedure *Greedy*:

1. Let $M'_{1l} = M_{1l}$ and $M'_{1m} = M_{1m}$. Sort items in M'_{1l} in non-increasing order of their sizes.
2. If M'_{1l} or M'_{1m} is empty then go to 5.
3. Let u_i be the first item in M'_{1l}.
4. Let u_j be the first item in M'_{1m} satisfying $w_i + w_j \leq 1$. If it does not exist, then delete item u_i from set M'_{1l} and go to 2; Otherwise pack items u_i and u_j into a new bin, delete them from set M'_{1l} and M'_{1m} respectively. Go to 2.
5. For all unpacked medium items, pack them pairwise into a bin. If the number of unpacked medium items is odd, then pack the last unpacked medium item alone into a bin.
6. For all unpacked large items with penalty at least 1, pack them each into a bin.
7. Pack huge items each into a bin.

Call the bins used in Steps 4-7 as B_1-, B_2-, B_3-, and B_4-*bins*, respectively. In the remainder of this subsection, we call a bin *open* if it is allowed to pack more items, otherwise it is *closed*. Now we can describe the algorithm.

Algorithm $RFF4$:

1. Apply procedure *Greedy*. Sort items in M_{1s} in non-increasing order of their sizes. Let all B_1-, B_2-, and B_4-bins be closed.
2. Arbitrarily choose an open B_3-bin if it exists, else go to 5.
3. Pack the first small item in M_{1s} into this open bin, if it fits into this bin, and delete it from M_{1s}. Go to 3.
4. If the first small item in M_{1s} does not fit into this open bin, close this bin, and go to 2.
5. If there does not exist open B_3-bin, (and M_{1s} is not empty yet), and there is at least one unpacked large item in M_1, then pack the first unpacked large item into a new bin. Call this bin as an *open* B_5*-bin*. Go to 3.
6. If there does not exist unpacked small item, and there exist unpacked large items, then reject them. Denote by R_6 the set consisting of all rejected large items. Go to 8.
7. If all large items are packed, then pack all remaining small items in M_{1s} into new bins by algorithm FF. Denote these bins by B_7-bins. Go to 8.
8. Reject all items in M_2.

Theorem 6. $R^{\infty}_{RFF4} \leq 3/2$.

3 The Problem BCR

3.1 Online Algorithm

Theorem 7. *No online algorithm can have an absolute competitive ratio of greater than* 0.

Hence in the remainder of this subsection, we consider the asymptotic competitive ratios of online algorithms. Since for the bin covering problem, no online algorithm can have an asymptotic competitive ratio of greater than 1/2 [4], it is still true for BCR. Next we present an online algorithm with a matching asymptotic competitive ratio of 1/2.

Noting that *Dual Next Fit* (DNF) is the best possible online algorithm for the classical bin covering problem in terms of the asymptotic competitive ratio [1], we next show that a simple modification of DNF also works for BCR.

Algorithm $MDNF$:

1. If the incoming item is in M_2, pack it by DNF algorithm.
2. If the incoming item is in M_1, reject it.

Lemma 3. *Let I' be an instance of the classical bin covering with n' items.* (1) ([1]) *We have $DNF(I') \geq \lfloor \frac{\sum_{j=1}^{n'} w_j}{2} \rfloor$.* (2) *Furthermore, if $\sum_{j=1}^{n'} w_j \geq 2k+1$ for some integer k, then $DNF(I') \geq k+1$.*

Theorem 8. *$R_{MDNF}^{\infty} = 1/2$, thus $MDNF$ is the best possible online algorithm in terms of the asymptotic competitive ratio.*

3.2 Offline Algorithm

Since the classical bin covering problem is a special case of BCR, we conclude that no offline algorithm can have an absolute worst-case ratio of greater than 1/2, unless $P = NP$ ([1]). Next we present a *modified Dual Next Fit Decreasing (MDNFD)* algorithm with an absolute worst-case ratio of 1/2.

Algorithm $MDNFD$:

1. If $\sum_{i=1}^{n} w_i < 1$, reject all items and stop.
2. If $W(M_2) < 1$ and $P(M_1) < 1$, pack all items into one bin and stop.
3. If $W(M_2) < 1$ and $P(M_1) \geq 1$, reject all items and stop.
4. Determine an integer $\omega \geq 0$ and real number $0 \leq \gamma < 2$ such that $W(M_2) = 2\omega + 1 + \gamma$.
5. If $\gamma \leq 1$ or $P(M_1) \geq 1$, then apply algorithm $MDNF$ to all items and stop.
6. If $W(M_2 \cup M_1) \geq 2\omega + 3$, then accept all items, and pack them by DNF algorithm. Stop.
7. Sort the items of M_2 in non-increasing order of their ratios between sizes and profits w_i/p_i such that $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \cdots \geq \frac{w_{|M_2|}}{p_{|M_2|}}$. Determine $\bar{k} = \min\{j : \sum_{i=1}^{j} w_i \geq 2\omega + 1, u_i \in M_2, i = 1, \cdots, j\}$.
8. Pack the first $\bar{k}$ items of M_2 by DNF, reject the remaining items of M_2 and all items of M_1. Stop.

Theorem 9. *$R_{MDNFD} = 1/2$, thus $MDNFD$ is the best possible in terms of the absolute worst-case ratio.*

The following instance can show that the asymptotic worst-case ratio is still 1/2. Let I be an instance with $2N + 2$ items, each with size $1 - \epsilon$ and profit $1 - 2\epsilon$. Then we know $M_1 = \emptyset$. $MDNFD$ stops at Step 5, and we have $\frac{MDNFD(I)}{OPT(I)} = \frac{N+1}{(2N+2)(1-2\epsilon)} \to \frac{1}{2}$ $(\epsilon \to 0)$. Therefore, to give an offline algorithm with an asymptotic worst-case ratio of greater than 1/2 is open.

References

1. S. B. Assman, D. S. Johnson, D. J. Kleitman, J. Y. T. Leung, On the dual of one-dimensional bin-packing problem, *J. of Algorithms*, 5(1984) 502-525.
2. E. G. Coffman, M. R. Garey, D.S. Johnson, Approximation algorithms for bin packing: A survey, In: D. Hochbaum eds., Approximation algorithms for NP-hard problems, PWS Publishing, 1997, 46-93.
3. J. Csirik, D. S. Johnson, C. Kenyon, Better approximation algorithms for bin covering, *Proc. of SIAM Conference on Discrete Algorithms*, Washington, DC, 2001, 557-566.
4. J. Csirik, V. Totik, On-line algorithms for a dual version of bin packing, *Discrete Applied Mathematics*, 21 (1988) 163-167.
5. J. Csirik, G. Woeginger, On-line packing and covering problems, *Lecture Notes in Computer Science 1442*, Berlin, Springer, 1998, 147-177.
6. W. Chen, J. Yang, D. Lu, Y. Pan, Two mathematical models and algorithms of internet communications, *Chinese J. Computers*, 21 (1999) 51-55. (in Chinese)
7. M. R. Garey, D. S. Johnson, Computer and Intractability: A Guide to the theory of NP-Completeness, New York, Freeman, 1979.
8. K. Jansen, R. Solis-Oba, An asymptotic fully polynomial time approximation scheme for bin covering, *Theoretical Computer Science*, 306(2003) 543-551.
9. D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, R. L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms, *SIAM J. on Computing*, 3(1974) 299-325.
10. N. Karmarkar, R. M. Karp, An efficient approximation scheme for the one-dimensional bin packing problem, *Proc. 23rd Annual Symposium on Foundations of Computer Science*, Chicago, 1982, 312–320.
11. S. S. Seiden, On the online bin packing problem, *J. of the ACM*, 49(2002) 640-671.
12. D. Simchi-Levi, New worst-case results for the bin-packing problemma, *Naval Research Logistics*, 41(1994), 579-585.
13. A. Van Vliet, An improved lower bound for on-line bin packing algorithms, *Information Processing Letters*, 43(1992) 277-284.
14. D. Ye, G. Zhang, On-line scheduling of parallel jobs, *Lecture Notes in Computer Science 3104*, Berlin, Springer, 2004, 279-290.
15. G. C. Zhang, X. Q. Cai, C. K. Wong, Linear-time approximation algorithms for bin packing problem, *Operations Research Letters*, 26(1999) 217-222.

Query-Monotonic Turing Reductions*

Lane A. Hemaspaandra[1] and Mayur Thakur[2,**]

[1] Department of Computer Science, University of Rochester, Rochester, NY 14627, USA
lane@cs.rochester.edu
[2] Department of Computer Science, University of Missouri–Rolla, Rolla, MO 65409, USA
thakurk@umr.edu

Abstract. We study reductions that limit the extreme adaptivity of Turing reductions. In particular, we study reductions that make a rapid, structured progression through the set to which they are reducing: Each query is strictly longer (shorter) than the previous one. We call these reductions query-increasing (query-decreasing) Turing reductions. We also study query-nonincreasing (query-nondecreasing) Turing reductions. These are Turing reductions in which the sequence of query lengths is nonincreasing (nondecreasing). We ask whether these restrictions in fact limit the power of reductions. We prove that query-increasing and query-decreasing Turing reductions are incomparable with (that is, are neither strictly stronger than nor strictly weaker than) truth-table reductions and are strictly weaker than Turing reductions. In addition, we prove that query-nonincreasing and query-nondecreasing Turing reductions are strictly stronger than truth-table reductions and strictly weaker than Turing reductions. Despite the fact that we prove query-increasing and query-decreasing Turing reductions to in the general case be strictly weaker than Turing reductions, we identify a broad class of sets A for which any set that Turing reduces to A will also reduce to A via both query-increasing and query-decreasing Turing reductions. In particular, this holds for all *tight paddable* sets, where a set is said to be tight paddable exactly if it is paddable via a function whose output length is bounded tightly both from above and from below in the length of the input. We prove that many natural NP-complete problems such as satisfiability, clique, and vertex cover are tight paddable.

1 Introduction

Oracle access is an important notion in the theory of computation. It forms the basis for defining the different levels of the polynomial hierarchy [16, 19]. It is central to the notion of Turing reducibility [14], which is used to compare the complexity of problems.

How the *nature* of access to its oracle affects the power of a Turing machine is a central research issue. For example, the issue of whether adaptive access is more powerful than nonadaptive access has been well researched (see, e.g., [1, 18, 20])[1]. A Turing

* Supported in part by grant NSF-CCF-0426761

** Work done in part while at the University of Rochester

[1] More broadly, the differences between various polynomial-time reducibilities have been extensively studied ever since the seminal work of Ladner, Lynch, and Selman [14]. For an overview of this line of research, we refer the reader to the survey articles by Homer [11, 12], Buhrman and Torenvliet [3], and Pavan [17]

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 895–904, 2005.

machine with adaptive access to its oracle has the full flexibility of asking queries in any order. In particular, the kth query may depend on the answers to the previous $k-1$ queries. On the other hand, in nonadaptive access, the Turing machine is required to generate all queries before asking any of them. Thus, adaptive access and nonadaptive access might at first seem to represent two extremes of flexibility in deterministic oracle access (when one is without external limits on the number of queries allowed).

In this paper, we examine modes of oracle access whose flexibility in some cases lies somewhere between these two extremes, and in some cases is incomparable even to the lower "extreme," truth-table reducibility. To help us formalize the degree and nature of the flexibility of oracle access, we introduce the notion of *query-restricted* Turing reductions, $\leq^p_{\rho\text{-}T}$, where ρ is a set of query sequences. A Turing machine M has the ρ query property with respect to B if the following holds for each input x: If $q_1, q_2, \ldots, q_k$ is the sequence of strings queried by $M^B(x)$, then $(x, q_1, q_2, \ldots, q_k) \in \rho$. We say that $A \leq^p_{\rho\text{-}T} B$ if there is a deterministic Turing machine M such that M robustly (i.e., for all oracles) runs in polynomial time, $L(M^B) = A$, and M has the ρ query property with respect to B.

Note that each query-restricted Turing reduction $\leq^p_{\rho\text{-}T}$ imposes certain restrictions (formally captured by ρ) on the sequence of strings queried by the machine. In this paper, we study query-restricted Turing reductions in which the set of allowable query sequences imposes monotonicity in the *length* of the queries that the underlying machine asks to the oracle. We call such reductions *query-monotonic Turing reductions*. For example, in *query-increasing* Turing reductions, the machine is required to ask its queries in a length-increasing fashion, i.e., each query must be longer than all the earlier ones (and also, in its strongest form, longer than the input). The main query-monotonic reductions that we study in this paper are query-increasing reductions ($\leq^p_{li\text{-}T}$), query-decreasing reductions ($\leq^p_{ld\text{-}T}$), query-nonincreasing reductions ($\leq^p_{lni\text{-}T}$), and query-nondecreasing reductions ($\leq^p_{lnd\text{-}T}$). (Formal definitions of these and other query-monotonic reductions are provided in Section 2.)

We came to define the notion of query-monotonicity motivated by the idea of rapid progress. In particular, in this paper we wish to study the power of polynomial-time machines that sweep, directionally, through their database (i.e., oracle). Such machines will have a "use it when you can" flavor to their access to the information at each length – once one query at or beyond a length is asked, the rest of the information at that length is forever lost to direct access, on the current input. However, this apparently restrictive access is not necessarily restrictive in effect. It is at least plausible that, by exploiting properties of particular databases, the restriction (for those databases) will be toothless, i.e., can be obeyed without loss of generality. In this paper, we show that in some cases the restriction has teeth, but we also show that in other quite central cases the restriction is toothless. In particular, for several reductions $\leq^p_\alpha$ and $\leq^p_\beta$, we ask whether, for all $A, B \subseteq \Sigma^*$, $A \leq^p_\alpha B$ implies $A \leq^p_\beta B$. If the answer to this question is "true", we say that $\leq^p_\beta$ is stronger than $\leq^p_\alpha$. (Note that "stronger" does not in this paper necessarily promise "strictly stronger" – for example, each reduction is, under our definition, stronger than itself, but obviously is not strictly stronger than itself. Similarly, "weaker" does not necessarily promise "strictly weaker.") Roughly speaking, we show that if $\leq^p_\alpha$ and $\leq^p_\beta$ are chosen from among the Turing reductions, truth-table

reductions, 2-truth-table reductions, and the set of query-monotonic reductions that we study, then the only "is stronger than" relationships that hold are the ones that obviously follow directly from the definitions. The rest provably do not hold. For example, a query-increasing reduction is by definition also a query-nondecreasing reduction (since each length-increasing query sequence is also a length-nondecreasing query sequence). Thus, query-nondecreasing reductions are stronger than query-increasing reductions. We prove that the converse is not true. That is, we prove that query-increasing reductions are strictly weaker than query-nondecreasing reductions.

It is clear from the definitions that query-monotonic Turing reductions are no less restrictive than Turing reductions. Furthermore, from our results mentioned above it is clear that in some cases the query-increasing and query-decreasing restrictions have teeth. Thus, it is interesting to ask whether there are cases in which the restriction is toothless. In particular, we ask the following question: What structural properties should a set S have so that each set that is Turing reducible to S is in fact also query-increasing (query-decreasing) Turing reducible to S? We show that for a large class of sets – namely those that are paddable via a (polynomial-time) function whose output size is bounded tightly both from above and from below in the length of the input – the query-increasing (query-decreasing) restriction does not limit the power of polynomial-time oracle Turing machines using these sets as their database. We call these sets *tight paddable*. On one hand, we prove that there are NP-complete sets that are not tight paddable, but on the other hand hand, we show that many natural NP-complete problems (for example, satisfiability, vertex cover, and maximum clique) are tight paddable.

Glaßer et al. [6] have recently shown that all NP-complete sets are many-one autoreducible. This is a very powerful result. However, our results neither seem to be implied by nor seem to imply theirs.

This paper is organized as follows. Sections 2 and 3 define the different query-monotonic reductions that are used in this paper, and show that all these reductions are robust in a certain sense. In Section 4, we compare the power of query-monotonic reductions with that of Turing and truth-table reductions. In Section 5, we study the query-monotonic reduction closures of sets in NP and prove that many important NP-complete sets are tight paddable.

(Due to space limitations, all proofs, almost all of the discussion in the results sections, and some results are omitted. These in general can or will be found in the current technical report version of the paper [10] or in the in-preparation revised, extended version of that report.)

2 Preliminaries

The alphabet for all strings, unless otherwise mentioned, is $\Sigma = \{0, 1\}$. The length of a string x is denoted by $|x|$.

We use the standard Turing machine model as described, for example, in [13]. We view all Turing machines as potentially taking an oracle (or a pair of oracles), and so write "Turing machine" rather than "oracle Turing machine." Let M_1, M_2, ... be a standard enumeration of deterministic polynomial-time Turing machines such that the running time of M_i is bounded by $n^i + i$. Note that we build into our requirements

of the standard enumeration the fact that Turing machines in the enumeration defined above run in time that is bounded by a polynomial that is *independent* of the oracle(s) attached to the machine[2]. Thus, when we say that M is a polynomial-time Turing machine, we mean that there is a $k \in \mathbb{N}$ such that, for each $X \subseteq \Sigma^*$, each $Y \subseteq \Sigma^*$, and for each $x \in \Sigma^*$, $M^{X,Y}(x)$ runs in time $|x|^k + k$ (thus so does $M^X(x)$ "=" $M^{X,\emptyset}(x)$ and $M(x)$ "=" $M^{\emptyset,\emptyset}(x)$). (For each Turing machine M and for each $Y, Z \subseteq \Sigma^*$, $M^{Y,Z}$ represents machine M with Y as its first oracle and Z as its second oracle. See [8, 9] for earlier uses of double oracles – there in the context of studying the effect of the order in which the two databases are accessed.)

For any Turing machine N and any $x \in \Sigma^*$, we will use $N(x)$ as an abbreviation for "the computation of N on x." We will use DPTM as an abbreviation for "deterministic polynomial-time Turing machine." We use the standard meanings and notations for polynomial-time many-one reduction ($\leq^p_m$), polynomial-time Turing reduction ($\leq^p_T$), polynomial-time truth-table reduction ($\leq^p_{tt}$), and for each $k \in \mathbb{N}$, polynomial-time k-truth-table reduction ($\leq^p_{k\text{-}tt}$) [14].

We now introduce query-monotonic reductions. We in fact define a general "query-restricted" Turing reduction, $\leq^p_{\rho\text{-}T}$, where ρ is a restriction on the allowed sequences. Query-monotonic reductions are defined in terms of query-restricted Turing reductions: Each query-monotonic reduction is a query-restricted Turing reduction for some particular restriction ρ.

Definition 1. *Let* $\rho \subseteq \Sigma^* \cup \Sigma^* \times \Sigma^* \cup \Sigma^* \times \Sigma^* \times \Sigma^* \cup \cdots$.

1. *For each Turing machine* M*, each* $B \subseteq \Sigma^*$*, and each* $x \in \Sigma^*$*, we say that* M has the ρ query property with respect to B on input x *if the sequence* $q_1, q_2, \ldots, q_k$ *of queries made by* $M^B(x)$ *to its oracle satisfies:* $(x, q_1, q_2, \ldots, q_k) \in \rho$*. (For example,* $M^B(x)$ *may legally ask no queries only if* $(x) \in \rho$*.)*
2. *For each Turing machine* M *and each* $B \subseteq \Sigma^*$*, we say that* M has the ρ query property with respect to B *if, for each* $x \in \Sigma^*$*,* M *has the* ρ *query property with respect to* B *on input* x*.*
3. $A \leq^p_{\rho\text{-}T} B$ *if there exists a* DPTM M *such that*
   (a) M *has the* ρ *query property with respect to* B*, and*
   (b) $L(M^B) = A$*.*

We now define several query-monotonic restrictions that will be of interest to us. Note that the "Σ^*" part in the following definitions makes it legal for a Turing machine to ask no queries to its oracle.

Definition 2. *1.* (Length-increasing) $\rho_{li} = \Sigma^* \cup \{(x, q_1, q_2, \ldots, q_k) \mid k \geq 1 \wedge |q_1| < |q_2| < \ldots < |q_k|\}$.
2. (Length-decreasing) $\rho_{ld} = \Sigma^* \cup \{(x, q_1, q_2, \ldots, q_k) \mid k \geq 1 \wedge |q_1| > |q_2| > \ldots > |q_k|\}$.

[2] This is certainly not true of all Turing machines. There has been some study of the difference between the power of machines that have a robust – that is, independent of the oracle – polynomial-time bound on their running time and those that run in polynomial-time for each oracle yet have no robust polynomial-time bound on their running time [4]

3. (Length-nonincreasing) $\rho_{lni} = \Sigma^* \cup \{(x, q_1, q_2, \ldots, q_k) \mid k \geq 1 \wedge |q_1| \geq |q_2| \geq \ldots \geq |q_k|\}$.
4. (Length-nondecreasing) $\rho_{lnd} = \Sigma^* \cup \{(x, q_1, q_2, \ldots, q_k) \mid k \geq 1 \wedge |q_1| \leq |q_2| \leq \ldots \leq |q_k|\}$.
5. (Strong length-increasing) $\rho_{s\text{-}li} = \Sigma^* \cup \{(x, q_1, q_2, \ldots, q_k) \mid k \geq 1 \wedge |x| < |q_1| < |q_2| < \ldots < |q_k|\}$.
6. (Strong length-decreasing) $\rho_{s\text{-}ld} = \Sigma^* \cup \{(x, q_1, q_2, \ldots, q_k) \mid k \geq 1 \wedge |x| > |q_1| > |q_2| > \ldots > |q_k|\}$.
7. (Strong length-nonincreasing) $\rho_{s\text{-}lni} = \Sigma^* \cup \{(x, q_1, q_2, \ldots, q_k) \mid k \geq 1 \wedge |x| \geq |q_1| \geq |q_2| \geq \ldots \geq |q_k|\}$.
8. (Strong length-nondecreasing) $\rho_{s\text{-}lnd} = \Sigma^* \cup \{(x, q_1, q_2, \ldots, q_k) \mid k \geq 1 \wedge |x| \leq |q_1| \leq |q_2| \leq \ldots \leq |q_k|\}$.

For each $\alpha \in \{li, ld, lnd, lni, s\text{-}li, s\text{-}ld, s\text{-}lni, s\text{-}lnd\}$, and for each $A, B \subseteq \Sigma^*$, we will abuse notation and use $A \leq^p_{\alpha\text{-}T} B$ when we formally mean $A \leq^p_{\rho_\alpha\text{-}T} B$. For each Turing machine M and each $B \subseteq \Sigma^*$, if M has the ρ_{li} (respectively, ρ_{ld}, ρ_{lni}, ρ_{lnd}, $\rho_{s\text{-}li}$, $\rho_{s\text{-}ld}$, $\rho_{s\text{-}lni}$, $\rho_{s\text{-}lnd}$) query property with respect to B, then we say that M *has the query-increasing* (respectively, *query-decreasing*, *query-nonincreasing*, *query-nondecreasing*, *strong query-increasing*, *strong query-decreasing*, *strong query-nonincreasing*, *strong query-nondecreasing*) *property with respect to* B. If $A \leq^p_{li\text{-}T} B$ (respectively, $A \leq^p_{ld\text{-}T} B$, $A \leq^p_{lni\text{-}T} B$, $A \leq^p_{lnd\text{-}T} B$, $A \leq^p_{s\text{-}li\text{-}T} B$, $A \leq^p_{s\text{-}ld\text{-}T} B$, $A \leq^p_{s\text{-}lni\text{-}T} B$, $A \leq^p_{s\text{-}lnd\text{-}T} B$), then we say that A *polynomial-time query-increasing* (respectively, *query-decreasing*, *query-nonincreasing*, *query-nondecreasing*, *strong query-increasing*, *strong query-decreasing*, *strong query-nonincreasing*, *strong query-nondecreasing*) *Turing reduces to* B.

We now define relativized query-monotonic reductions. For each $\rho \subseteq \Sigma^* \cup \Sigma^* \times \Sigma^* \cup \Sigma^* \times \Sigma^* \times \Sigma^* \cup \cdots$, we say that, *relative to* Z, M *has the* ρ *query property with respect to* Y if, for each $x \in \Sigma^*$, the sequence $q_1, q_2, \ldots, q_k$ of queries that $M^{Y,Z}(x)$ asks to its first oracle satisfies $(x, q_1, q_2, \ldots, q_k) \in \rho$. $A \leq^{p,C}_{\rho\text{-}T} B$ if there exists a DPTM M such that $L(M^{B,C}) = A$ and, relative to C, M has the ρ query property with respect to B.

We now define a new notion of padding – *tight paddability* – that we will use to understand query-monotonic reductions over NP. Before we define tight paddability, for comparison we review notions of padding that are standard in the literature. Paddability of sets has been used in complexity theory in many contexts including the seminal work of Berman and Hartmanis [2] on polynomial-time isomorphism for sets in NP, and of Hartmanis [7] on the study of logspace isomorphism for sets in NL, CSL, P, NP, PSPACE, etc. Mahaney and Young [15] use padding functions to prove results regarding the structure of polynomial-time many-one degrees.

A (polynomial-time) paddable set is one for which there is a polynomial-time function (called a padding function for that set) that allows one to map the input string to a longer string while preserving membership in the set.

Definition 3. *Let* $A \subseteq \Sigma^*$. *Then* $\sigma : \Sigma^* \to \Sigma^*$ *is a* padding function *for* A *if*

1. σ *is polynomial-time computable,*
2. *for each* $x \in \Sigma^*$, $\sigma(x) \in A$ *if and only if* $x \in A$, *and*
3. *for each* $x \in \Sigma^*$, $|\sigma(x)| > |x|$.

We say that A is paddable *if there exists a padding function for* A.

Berman and Hartmanis [2] define two different types of functions called Z-padding functions and S-padding functions (though, formally speaking S-padding functions need not be padding functions in the sense of Definition 3).

Definition 4. *[2] Let* $A \subseteq \Sigma^*$. *Then* $\sigma : \Sigma^* \to \Sigma^*$ *is a* Z-padding function *for* A *if* σ *is a padding function and* σ *is a one-to-one function. We say that* A is Z-paddable *if there exists a Z-padding function for* A.

Definition 5. *[2] Let* $A \subseteq \Sigma^*$. *Then* $\sigma : \Sigma^* \times \Sigma^* \to \Sigma^*$ *is an* S-padding function *for* A *if*

1. σ *is polynomial-time computable,*
2. σ *is polynomial-time invertible in the second argument, i.e., there is a polynomial-time computable function* σ' *such that, for each* $x, y \in \Sigma^*$, $\sigma'(\sigma(x, y)) = y$, *and*
3. *for each* $x, y \in \Sigma^*$, $\sigma(x, y) \in A$ *if and only if* $x \in A$.

We say that A is S-paddable *if there exists an S-padding function for* A.

Mahaney and Young [15] note that any set that is S-paddable is S-paddable via some function that is polynomial-time invertible in both arguments, i.e., there exists a polynomial-time computable function ρ such that for each x and y, $\rho(\sigma(x, y)) = \langle x, y \rangle$. Also, note that if a function is invertible in both arguments, in the particular sense just mentioned, then it is one-to-one. We can use this property of S-paddable sets to show that each S-paddable set is also Z-paddable. Note that most natural NP-complete problems (for example, SAT, vertex cover, max clique, 3-colorability, etc.) are obviously S-paddable.

We now define two notions of tight padding: *tight paddability* and *tight Z-paddability*. Informally speaking, a tight padding function for a set is a padding function that has strong guarantees on the length of its output.

Definition 6. *Let* $A \subseteq \Sigma^*$.

1. *Let* $\sigma : \Sigma^* \to \Sigma^*$. *Then* σ is a tight padding function for A *if* σ *is a padding function for* A *and there exists a* $k \in \mathbb{N}$ *such that, for each* $x \in \Sigma^*$, $|\sigma(x)| \leq |x|+k$. *We say that* A is tight paddable *if there is a tight padding function for* A.
2. *Let* $\sigma : \Sigma^* \to \Sigma^*$. *Then* σ is a tight Z-padding function for A *if* σ *is a Z-padding function for* A *and there exists a* $k \in \mathbb{N}$ *such that, for each* $x \in \Sigma^*$, $|\sigma(x)| \leq |x|+k$. *We say that* A is tight Z-paddable *if there is a tight Z-padding function for* A.

Clearly, each tight paddable set is also paddable. Similarly, each tight Z-paddable set is also Z-paddable. Figure 1 shows the relationships among the different notions of paddability.

One might ask why we have not defined a "tight" analog of the S-paddable sets. One could. But we mention in passing that no one-to-one 2-ary function satisfies the length restriction associated with tightness. Let σ be an arbitrary one-to-one 2-ary function. Let $n \in \mathbb{N}$. The number of pairs of strings x and y such that $|x| + |y|$ is n is exactly $(n+1)2^n$. Thus, the length of the longest string whose preimage (x, y) in σ is such that

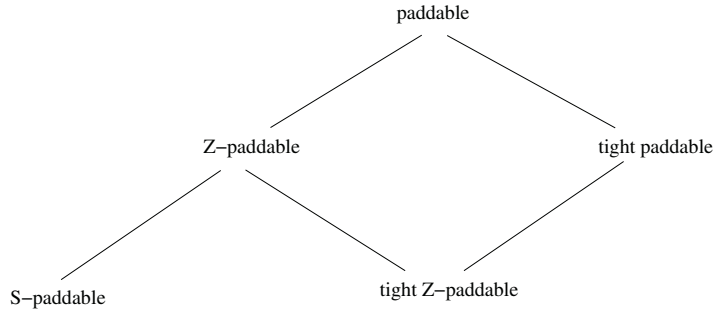


Fig. 1. Relationships between different notions of paddability. A line between a and b such that a lies above b indicates that any set of type b is also a set of type a

$|x| + |y| = n$ is at least $\lceil \log(1 + (n+1)2^n) - 1 \rceil \geq n - 1 + \log(n+1)$. Thus, for any such σ and for each $k \in \mathbb{N}$, there exist x and y such that $\sigma(x, y) > |x| + |y| + k$.

In Section 5, we show that many well-known NP-complete problems such as satisfiability of boolean formulas, vertex cover in graphs, and size of the largest clique in graphs are tight Z-paddable. We also show that many types of query-monotonic reductions are equivalent (and in fact, also equivalent to Turing reductions) when the set being reduced to is tight paddable.

3 Robustness of Query-Monotonic Reductions

In this section, we study the question: Given $A \leq^p_{\rho\text{-}T} B$, under what conditions is this $\leq^p_{\rho\text{-}T}$ reduction witnessed by a machine that has the robust (that is, with respect to all oracles) ρ query property?

Definition 7. *Let $\rho \subseteq \Sigma^* \cup \Sigma^* \times \Sigma^* \cup \Sigma^* \times \Sigma^* \times \Sigma^* \cup \cdots$.*

1. *For each Turing machine M, we say that M* has the robust ρ query property *if, for each $C \subseteq \Sigma^*$, M has the ρ query property with respect to C.*
2. *For each $A, B \subseteq \Sigma^*$, $A \leq^p_{r\text{-}\rho\text{-}T} B$ if there exists a* DPTM *M such that*
   (a) M has the robust ρ query property, and
   (b) $L(M^B) = A$.

Theorem 8. *Let $\rho \subseteq \Sigma^* \cup \Sigma^* \times \Sigma^* \cup \Sigma^* \times \Sigma^* \times \Sigma^* \cup \cdots$ be a set of tuples such that $\rho' = \{\langle x_1, x_2, \ldots, x_k\rangle \mid (x_1, x_2, \ldots, x_k) \in \rho\} \in \mathrm{P}$ and ρ satisfies the following: For each $k \geq 0$, and each $x, q_1, q_2, \ldots, q_{k+1} \in \Sigma^*$, if $(x, q_1, x_2, \ldots, x_{k+1}) \in \rho$, then $(x_1, x_2, \ldots, x_k) \in \rho$. Then, for each $A, B \subseteq \Sigma^*$, $A \leq^p_{\rho\text{-}T} B$ if and only if $A \leq^p_{r\text{-}\rho\text{-}T} B$.*

Corollary 9. *For each $\alpha \in \{li, ld, lni, lnd, s\text{-}li, s\text{-}ld, s\text{-}lni, s\text{-}lnd\}$, and for each $A, B \subseteq \Sigma^*$, $A \leq^p_{\alpha\text{-}T} B$ if and only if $A \leq^p_{r\text{-}\alpha\text{-}T} B$.*

Theorem 10. *Let $\rho \subseteq \Sigma^* \cup \Sigma^* \times \Sigma^* \cup \Sigma^* \times \Sigma^* \times \Sigma^* \cup \cdots$ be a set of tuples such that the set $\rho' = \{\langle x_1, x_2, \ldots, x_k\rangle \mid (x_1, x_2, \ldots, x_k) \in \rho\}$ is in* P. *If* $\mathrm{P} = \mathrm{NP}$, *then for each $A, B \subseteq \Sigma^*$, $A \leq^p_{\rho\text{-}T} B$ if and only if $A \leq^p_{r\text{-}\rho\text{-}T} B$.*

Theorem 11. *There is an oracle C, a set B, and a constraint set ρ such that the set $\rho' = \{\langle x_1, x_2, \ldots, x_k\rangle \mid (x_1, x_2, \ldots, x_k) \in \rho\} \in \mathrm{P}^C$, $\emptyset \leq^{p,C}_{\rho\text{-}T} B$, and yet $\emptyset \not\leq^{p,C}_{r\text{-}\rho\text{-}T} B$.*

4 Comparing Query-Monotonic Reductions to Turing and Truth-Table Reductions

In this section, we compare the power of query-monotonic reductions to different forms of Turing and truth-table reductions. By definition, query-monotonic reductions are no more powerful than Turing reductions. But in which cases are they as powerful as Turing reductions and in which cases are they not? How does the power of query-monotonic reductions relate to the that of classical reductions such as Turing reductions and truth-table reductions? These are the central questions that we answer in this section.

Let $\leq^p_\alpha$ and $\leq^p_\beta$ be defined reductions. If $(\forall A, B \subseteq \Sigma^*)[A \leq^p_\alpha B \implies A \leq^p_\beta B]$, then we say that $\leq^p_\beta$ *is stronger than* $\leq^p_\alpha$ and that $\leq^p_\alpha$ *is weaker than* $\leq^p_\beta$. If $\leq^p_\beta$ is stronger than $\leq^p_\alpha$ and $(\exists A, B \subseteq \Sigma^*)[A \leq^p_\beta B \wedge A \not\leq^p_\alpha B]$, then we say that $\leq^p_\beta$ *is strictly stronger than* $\leq^p_\alpha$ and that $\leq^p_\alpha$ *is strictly weaker than* $\leq^p_\beta$. (For example, $\leq^p_T$ is trivially stronger than itself, and is both stronger and strictly stronger than $\leq^p_{tt}$.)[3] If we say that $\leq^p_\beta$ is not stronger than $\leq^p_\alpha$, we mean just that – that "$\leq^p_\beta$ is stronger than $\leq^p_\alpha$" is untrue. That is, this means $(A, B \subseteq \Sigma^*)[A \leq^p_\alpha B \wedge A \not\leq^p_\beta B]$. Note of course that "$\leq^p_\beta$ is not stronger than $\leq^p_\alpha$" and "$\leq^p_\beta$ is strictly weaker than $\leq^p_\alpha$" are not synonymous. Also note that "$\leq^p_\beta$ is strictly stronger than $\leq^p_\alpha$" is equivalent to "$\leq^p_\beta$ is stronger than $\leq^p_\alpha$ but $\leq^p_\alpha$ is not stronger than $\leq^p_\beta$."

Let $X = \{li, ld, lni, lnd, s\text{-}li, s\text{-}ld, s\text{-}lni, s\text{-}lnd\}$. For each $\alpha, \beta \in X$, we ask whether $\leq^p_\alpha$ is stronger than $\leq^p_\beta$. There are 64 ($8 \times 8 = 64$) such questions. We resolve all these questions. In fact, we show that the only "stronger than" relationships that hold are the ones that immediately follow from the definitions. Since the definitions of query-monotonic reductions are based on restriction sets (see Definition 2), we can formally (yet succinctly) state the answers to each of these 64 questions as Theorem 12.

Theorem 12. *For each $\alpha, \beta \in \{li, ld, lni, lnd, s\text{-}li, s\text{-}ld, s\text{-}lni, s\text{-}lnd\}$, $\leq^p_\beta$ is stronger than $\leq^p_\alpha$ if and only if $\rho_\alpha \subseteq \rho_\beta$.*

We now compare query-monotonic reductions to 2-truth-table, truth-table, and Turing reductions.

Theorem 13. $(\exists A, E \subseteq \Sigma^*)[E \leq^p_T A \wedge E \not\leq^p_{lni\text{-}T} A \wedge E \not\leq^p_{lnd\text{-}T} A]$.

Corollary 14. $(\exists A, E \subseteq \Sigma^*)[E \leq^p_T A \wedge E \not\leq^p_{li\text{-}T} A \wedge E \not\leq^p_{ld\text{-}T} A]$.

Theorem 15. $(\exists A, L_A \subseteq \Sigma^*)[L_A \leq^p_{2\text{-}tt} A \wedge L_A \not\leq^p_{li\text{-}T} A \wedge L_A \not\leq^p_{ld\text{-}T} A]$.

Theorem 16. $(\exists A, L_A \subseteq \Sigma^*)[L_A \leq^p_{s\text{-}li\text{-}T} A \wedge L_A \leq^p_{s\text{-}ld\text{-}T} A \wedge L_A \not\leq^p_{tt} A]$.

Theorem 17. *For each class $\mathcal{C}$ of languages such that $\mathcal{C}$ is closed downward under $\leq^p_m$ reductions,* $\mathrm{R}^p_T(\mathcal{C}) = \mathrm{R}^p_{lni\text{-}T}(\mathcal{C}) = \mathrm{R}^p_{lnd\text{-}T}(\mathcal{C})$

[3] So, as is now standard in complexity, we use "stronger" to refer to reductions that link at least as many sets as the ones they are stronger than. To avoid confusion, we mention in passing that recursion theorists sometimes exchange "strong" and "weak," as they focus on the level of refinement of the equivalence classes induced by the reductions, and a few computer science papers – especially early ones – mirrored that notion

Corollary 18. *For each class* $\mathcal{C} \in \{\mathrm{NP}, \mathrm{coNP}, \mathrm{BPP}, \mathrm{C_{=}P}, \oplus\mathrm{P}, \mathrm{PP}\}$, $\mathrm{R}^p_T(\mathcal{C}) = \mathrm{R}^p_{lni\text{-}T}(\mathcal{C}) = \mathrm{R}^p_{lnd\text{-}T}(\mathcal{C})$.

5 Query-Monotonic Reductions to NP Sets

In the previous section, we saw that query-monotonic and Turing reductions are different in general, though for many natural complexity classes the reduction closure of these classes with respect to query-nonincreasing, query-nondecreasing, and Turing reductions are identical. In this section, we study the relationship between query-monotonic and Turing reductions to NP sets. We ask the following question: For each $A \subseteq \Sigma^*$ and $B \in \mathrm{NP}$, does $A \leq^p_T B$ imply $A \leq^p_{li\text{-}T} B$? Since $\leq^p_T$ is stronger than $\leq^p_{li\text{-}T}$, if the answer to the above question is "yes," then for each $A \subseteq \Sigma^*$ and for $B \in \mathrm{NP}$, $A \leq^p_T B$ if and only if $A \leq^p_{li\text{-}T} B$. Similarly, we ask the following question: For each $A \subseteq \Sigma^*$ and each $B \in \mathrm{NP}$, does $A \leq^p_T B$ imply $A \leq^p_{ld\text{-}T} B$? Since $\leq^p_T$ is stronger than $\leq^p_{ld\text{-}T}$, if the answer to the above question is "yes," then for each $A \subseteq \Sigma^*$ and $B \in \mathrm{NP}$, $A \leq^p_T B$ if and only if $A \leq^p_{ld\text{-}T} B$. Both these issues are currently open.

Theorem 19. $(\exists W \subseteq \Sigma^*)(\exists A \in \mathrm{NP}^W)(\exists L_A \subseteq \Sigma^*)[L_A \leq^{p,W}_{2\text{-}tt} A \wedge L_A \not\leq^{p,W}_{li\text{-}T} A \wedge L_A \not\leq^{p,W}_{ld\text{-}T} A]$.

Hypothesis S: $(\forall A \subseteq \Sigma^*)(\forall B \in \mathrm{NP})[A \leq^p_T B \iff A \leq^p_{li\text{-}T} B]$.

Theorem 20. *There exists an infinite polynomial-time computable set* $L \subseteq \mathbb{N}$ *such that if Hypothesis S holds then, for each set* A *in* P^{NP}*, there is a* $C \in \mathrm{NP}$ *such that* $A \cap \{x \in \Sigma^* \mid |x| \in L\} \leq^p_{1\text{-}tt} C$.

Theorem 21. *Let* S *be a tight paddable set. Then, for each* $A \subseteq \Sigma^*$*, the following are equivalent:* $A \leq^p_T S$*,* $A \leq^p_{li\text{-}T} S$*,* $A \leq^p_{ld\text{-}T} S$*, and* $A \leq^p_{s\text{-}li\text{-}T} S$.

Theorem 22. *The following problems (see [5] for definitions of these problems) are tight Z-paddable (and thus, tight paddable):* 3-SAT, VERTEX COVER, CLIQUE, 3-COLORABILITY, MONOCHROMATIC TRIANGLE, BIPARTITE SUBGRAPH, MULTIPROCESSING SCHEDULING, CYCLIC ORDERING, QUADRATIC DIOPHANTINE EQUATIONS, MAX 2-SAT, *and* DECISION TREE.

Theorem 23. *If* $\mathrm{R}^p_T(\text{3-SAT}) = \mathrm{R}^p_{s\text{-}ld\text{-}T}(\text{3-SAT})$*, then* $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}[n]}$.

Theorem 24. *Each set, except* $\emptyset$ *and* Σ^**, is polynomial-time many-one equivalent to a set that is not tight paddable.*

Theorem 25. *Every set is polynomial-time many-one equivalent to a tight Z-paddable set.*

Acknowledgment

We thank Mitsunori Ogihara for helpful conversations.

References

1. K. Ambos-Spies and L. Bentzien. Separating NP-completeness notions under strong hypotheses. *Journal of Computer and System Sciences*, 61(3):335–361, 2000.
2. L. Berman and J. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322, 1977.
3. H. Buhrman and L. Torenvliet. On the structure of complete sets. In *Proceedings of the 9th Structure in Complexity Theory Conference*, pages 118–133. IEEE Computer Society Press, 1994.
4. J. Cai, L. Hemaspaandra, and G. Wechsung. Robust reductions. *Theory of Computing Systems*, 32(6):625–647, 1999.
5. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
6. C. Glaßer, M. Ogihara, A. Pavan, A. Selman, and L. Zhang. Autoreducibility, mitocity, and immunity. Technical Report TR 05-011, Electronic Colloquium on Computational Complexity, `http://www.eccc.uni-trier.de/eccc/`, 2005.
7. J. Hartmanis. On log-tape isomorphisms of complete sets. *Theoretical Computer Science*, 7(3):273–286, 1978.
8. E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. Query order in the polynomial hierarchy. *Journal of Universal Computer Science*, 4(6):574–588, 1998.
9. L. Hemaspaandra, H. Hempel, and G. Wechsung. Query order. *SIAM Journal on Computing*, 28(2):637–651, 1999.
10. L. Hemaspaandra and M. Thakur. Query-monotonic turing reductions. Technical Report TR-818, Department of Computer Science, University of Rochester, Rochester, NY, November 2003.
11. S. Homer. Structural properties of nondeterministic complete sets. In *Proceedings of the 5th Structure in Complexity Theory Conference*, pages 3–10. IEEE Computer Society Press, 1990.
12. S. Homer. Structural properties of complete problems for exponential time. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*. Springer-Verlag, 1997.
13. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
14. R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975.
15. S. Mahaney and P. Young. Reductions among polynomial isomorphism types. *Theoretical Computer Science*, 39(2–3):207–224, 1985.
16. A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, pages 125–129, October 1972.
17. A. Pavan. Comparison of reductions and completeness notions. *SIGACT News*, 34(2):27–41, 2003.
18. A. Pavan and A. Selman. Separation of NP-completeness notions. *SIAM Journal on Computing*, 31(3):906–918, 2002.
19. L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.
20. O. Watanabe. A comparison of polynomial time completeness notions. *Theoretical Computer Science*, 54(2–3):249–265, 1987.

On Sequential and 1-Deterministic P Systems*

Oscar H. Ibarra[1,**], Sara Woodworth[1], Hsu-Chun Yen[2], and Zhe Dang[3]

[1] Department of Computer Science
University of California, Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu
[2] Department of Electrical Engineering
National Taiwan University, Taipei, Taiwan 106, R.O.C.
[3] School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99164, USA

Abstract. The original definition of P-systems calls for rules to be applied in a maximally parallel fashion. However, in some cases a sequential model may be a more reasonable assumption. Here we study the computational power of different variants of sequential P-systems. Initially we look at cooperative systems operating on symbol objects and without prioritized rules, but which allow membrane dissolution and bounded creation rules. We show that they are equivalent to vector addition systems and, hence, nonuniversal. When these systems are used as language acceptors, they are equivalent to communicating P systems which, in turn, are equivalent to partially blind multicounter machines. In contrast, if such cooperative systems are allowed to create an unbounded number of new membranes (i.e., with unbounded membrane creation rules) during the course of the computation, then they become universal. We then consider systems with prioritized rules operating on symbol objects. We show two types of results: there are sequential P systems that are universal and sequential P systems that are nonuniversal. In particular, both communicating and cooperative P systems are universal, even if restricted to 1-deterministic systems with one membrane. However, the reachability problem for multi-membrane catalytic P systems with prioritized rules is NP-complete and, hence, these systems are nonuniversal.

1 Introduction

Initiated five years ago by Gheorghe Paun [17] as a branch of molecular computing, *membrane computing* identifies an unconventional computing model, namely a P system, from natural phenomena of cell evolutions and chemical reactions. A P system abstracts from the way the living cells process chemical compounds in their compartmental structure. Thus, regions defined by a membrane structure contain objects that evolve according to given rules. The objects can be described by symbols or by strings of symbols, in such a way that multisets of objects are placed in regions of the membrane structure. The membranes themselves are organized as a Venn diagram or a tree

* The work of Oscar H. Ibarra was supported in part by NSF Grants CCR-0208595 and CCF-0430945. The research of Hsu-Chun Yen was supported in part by NSC Grant 93-2213-E-002-003, Taiwan. The work of Zhe Dang was supported in part by NSF Grant CCF-0430531

** Corresponding author

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 905–914, 2005.

structure where one membrane may contain other membranes. By using the rules in a nondeterministic, maximally parallel manner, transitions between the system configurations can be obtained. A sequence of transitions shows how the system is evolving. Various ways of controlling the transfer of objects from a region to another and applying the rules, as well as possibilities to dissolve, divide or create membranes have been studied.

Membrane computing has been quite successful: many models have been introduced, most of them Turing complete and/or able to solve computationally intractable problems (NP-complete, PSPACE-complete) in a feasible time (polynomial), by trading space for time. (See the P system website at *http://psystems.disco.unimib.it* for a large collection of papers in the area, and in particular the monograph [18].) Due to the built-in nature of maximal parallelism inherent in the model, P systems have a great potential for implementing massively concurrent systems in an efficient way that would allow us to solve currently intractable problems (in much the same way as the promise of quantum and DNA computing) once future bio-technology (or silicon-technology) gives way to a practical bio-realization (or chip-realization). In fact, the Institute for Scientific Information (ISI) has recently selected membrane computing as a fast "Emerging Research Front" in Computer Science (*http://esi-topics.com/ erf/october2003.html*).

In the standard definition of a P system, the computation is carried out in a maximally parallel and nondeterministic manner [17, 18]. However, an interesting class of P systems with symport/antiport rules was studied in [5] where each system is *deterministic* in the sense that the computation path of the system is unique; i.e., at each step of the computation, the maximal multiset of rules that is applicable is unique. It was shown in [5] that any recursively enumerable unary language $L \subseteq o^*$ can be accepted by a deterministic 1-membrane symport/antiport system. Thus, for symport/antiport systems, the deterministic and nondeterministic versions are equivalent.

The construction of the deterministic system in [5] is such that the size of the maximal multiset of rules that is applicable at every step of the computation is either 1 or 2. We refer to this system as 2-deterministic. In general, a k-deterministic system is one in which the maximal multiset of rules applicable at each step is at most k. An interesting case is when $k = 1$, i.e., the system is 1-deterministic.

A concept, which is more general than 1-deterministic, is that of sequential mode of computation in P systems; i.e., at every step, only one nondeterministically chosen rule instance is applied. Clearly, when a P system is 1-deterministic, then the system (which, by definition, is still maximally parallel) can be treated as a sequential system. So if a class of systems is nonuniversal under the sequential mode, then any 1-deterministic such system in the class is also nonuniversal. Sequential P systems (also called asynchronous P systems) have been studied in various places in the literature (see, e.g., [1–4, 6, 11]). Here, we present results that complement these earlier results. In particular, we show the following:

1. Any sequential P system with cooperative rules (i.e., rules of the form $u \rightarrow v$, where u, v are strings of symbols) with rules for membrane creation and membrane dissolution can be simulated by a vector addition system (VAS), provided the rules are not prioritized and the number of membranes that can be created during the computation is bounded by some fixed positive integer. Hence the reachability

problem (deciding if a configuration is reachable from the start configuration) is decidable. It follows that 1-deterministic such systems have a decidable reachability problem. Interestingly and somewhat surprisingly, if such cooperative systems are allowed to create an unbounded number of new membranes during the course of the computation, then they become universal.

2. A sequential communicating P system language acceptor (CPA) is equivalent to a partially blind multicounter machine (PBCM) [7]. Several interesting corollaries follow from this equivalence, for example:
   (a) The emptiness problem for CPAs is decidable.
   (b) The class of CPA languages is a proper subclass of the recursive languages.
   (c) The language $\{a^n b^n \mid n \geq 1\}^*$ cannot be accepted by a CPA.
   (d) For every r, there is an $s > r$ and a language that can be accepted by a quasirealtime CPA with s membranes that cannot be accepted by a quasirealtime CPA with r membranes. (In a CPA, we do not assume that the CPA imports an input symbol from the environment at every step. Quasirealtime means that the CPA has to import an input symbol from the environment with delay of no more than k time steps for some nonnegative integer k independent of the computation.)
   (e) A quasirealtime CPA is strictly weaker than a linear time CPA. (Here, linear time means that the CPA accepts an input of length n within cn time for some constant c.)
   (f) The class of quasirealtime CPA languages is not closed under Kleene + and complementation.

   We note that the relationship between PBCMs and sequential symport/antiport P systems (similar to communication P systems) has been studied recently in [6], but only for systems with symbol objects and not as language acceptors. Thus, the results in [6] deal only with tuples of nonnegative integers defined by P systems and counter machines. For example, it was shown in [6] that a set of tuples of nonnegative integers that is definable by a partially blind counter machine can be defined by a sequential symport/antiport system with two membranes. Our new results above cannot be derived from the results in [6].
3. The results for CPA above generalize to cooperative system acceptors with membrane dissolution and bounded creation rules. Hence, the latter are also equivalent to PBCMs.
4. Any recursively enumerable unary language can be accepted by a 1-deterministic 1-membrane CPA with prioritized rules.
5. The reachability problem for sequential catalytic systems with prioritized rules (hence, for 1-deterministic such machines as well) is NP-complete. It follows from this result that a 1-deterministic catalytic system with prioritized rules can only accept recursive languages.

Note that from items 4 and 5 above, when the rules are prioritized, there are 1-deterministic systems that are universal and 1-deterministic systems that are not universal. In contrast, from item 1, without prioritized rules, 1-deterministic systems are not universal.

Due to page limitation, proofs are omitted. Complete proofs will appear later in an expanded (journal) version of this paper.

2 P Systems Without Prioritized Rules Operating in Sequential Mode

In this section, we consider the general definition of a P system as given originally by G. Paun in [17, 18], but with unprioritized rules. However, we allow rules for membrane dissolution and membrane creation. We look at systems that operate in sequential mode, i.e., exactly one rule instance is applied at each step (unless the system halts).

Before proceeding further, we need the definition of a vector addition system. An n-dimensional *vector addition system* (VAS) is a pair $G = \langle x, W\rangle$, where $x \in \mathbf{N}^n$ is called the *start point* (or *start vector*) and W is a finite set of vectors in $\mathbf{Z}^n$, where $\mathbf{Z}$ is the set of all integers (positive, negative, zero). The *reachability set* of the VAS $\langle x, W\rangle$ is the set $R(G) = \{z \mid$ for some j, $z = x + v_1 + ... + v_j$, where, for all $1 \leq i \leq j$, each $v_i \in W$ and $x + v_1 + ... + v_i \geq 0\}$. An n-dimensional *vector addition system with states* (VASS) is a VAS $\langle x, W\rangle$ together with a finite set T of transitions of the form $p \rightarrow (q, v)$, where p and q are states and v is in W. The meaning is that such a transition can be applied at point y in state p and yields the point $y + v$ in state q, provided that $y + v \geq 0$. The VASS is specified by $G = \langle x, W, T, p_0\rangle$, where p_0 is the starting state. It is known that n-dimensional VASS can be effectively simulated by $(n+3)$-dimensional VAS [8]. The *reachability problem* for a VAS (VASS) G is to determine, given a vector y, whether y is in $R(G)$. It is known that the reachability problem for VASS (and hence also for VAS) is decidable [13]. It is also known that VAS, VASS, and Petri net are all equivalent.

First, we study P systems with membrane dissolution but without rules for membrane creation. Clearly, for sequential computations, it is sufficient to only have cooperative rules of the form $u \rightarrow v$ or of the form $u \rightarrow v; \delta$, where u and v are strings of objects (symbols) where each symbol b in v has a designation or target, i.e., it is written b_x, where x can be $here$, out, or in_j. The designation $here$ means that the object b remains in the membrane containing it (we usually omit this target, when it is understood). The designation out means that the object is transported to the membrane directly enclosing the membrane that contains the object. The designation in_j means that the object is moved into a membrane, labeled j, that is directly enclosed by the membrane that contains the object. The δ attached to the rule means that after the application of the rule, the membrane and the rules it contains are dissolved and the objects in the membrane become part of the membrane that enclosed the dissolved membrane.

Theorem 1. *A sequential P system with unprioritized cooperative rules (possibly with membrane dissolution rules) can be simulated by a VASS. Hence its reachability problem is decidable.*

Another type of P systems (called *active P systems*) introduced in [15] allows rules to create new membranes during the computation. The objects of an active P system consist of *passive objects* which do not create new membranes and *active objects* which do create new membranes. A membrane creation rule is written as $u \rightarrow [_i v]_i$. If a rule of this form is applied in membrane r, this rule consumes u from r, creates a new membrane i within r, and creates the string v_{in_i}. Membrane creation changes how the degree of a system is defined. In a P system without membrane creation, its degree is the

number of membranes in the initial configuration. A P system which allows membrane creation must take into consideration the membranes that may be created. Hence, the degree of a system of this type is an ordered pair where the first component consists of the number of membranes in the initial configuration and the second component consists of the maximum number of membranes at any given time in the system during computation. The concept of membrane creation in a P system has a biological basis. In biology, reproduction is a fundamental function of most cells. Including this function in the definition of a P system is a natural generalization of the model. We have the following result:

Theorem 2. *A sequential P system with unprioritized rules and with both membrane dissolution and creation rules, where the total number of membranes that can be created during the computation is at most t (for some positive integer t), can be simulated by a VASS. Hence its reachability problem is decidable.*

The key reason why Theorem 2 works is the bound t of the total number of membranes created during any computation. What if we remove this condition? That is, suppose the number of times membrane creation rules are invoked during the computation is unbounded (i.e., it is a function of the computation). In this case, a sequential P system in Theorem 2 becomes universal. More precisely, we have the following:

Theorem 3. *A sequential P system with unprioritized rules and with both membrane dissolution and creation rules can simulate two-counter machines (and hence are universal).*

In the P system of Theorem 3, objects can move in and move out a membrane (e.g., the object of a state symbol). This move-in/out is essential in simulating the state transitions in a two-counter machine. We do not currently know whether the P systems in the theorem become nonuniversal when, in addition to membrane creation and dissolution rules, we only allow *local rules* in the form of $u \rightarrow v$ where the target of every object in v is $here$ (i.e., no move-in/out). This will be left as a topic for further investigation.

3 P Systems and Partially Blind Multicounter Machines

In a communicating P system CPS (with multiple membranes) [19], each rule is of one of the following forms: (1) $a \rightarrow a_x$, (2) $ab \rightarrow a_x b_y$, and (3) $ab \rightarrow a_x b_y c_{come}$, where a, b, c are objects, x, y (which indicate the directions of movements of a and b) can be $here$, out, or in_j (see Section 2 for their meanings). The $come$ can only occur within the outermost region (i.e., skin membrane), which brings in symbol c from the environment.

Here, we consider a variant of the CPS which is used as a language acceptor. Consider a CPS G with input alphabet $\Sigma = \{a_1, ..., a_k\}$. Let α be a new symbol not in Σ. We assume that only α and the symbols in Σ occur abundantly in the environment. Let $\Sigma_\alpha = \Sigma \cup \{\alpha\}$. Thus, the CPS can import an unbounded number of symbols in Σ_α from the environment. We assume that no symbol in Σ_α occurs in the initial configuration. We can view the CPS G above as a language acceptor, which we call a CPA. G accepts a string $x \in \Sigma^*$ if it constitutes all the symbols over Σ imported from the

environment during the computation, in the order given in x, when the system halts. Thus, x is built up as follows. At the start of the computation, $x = \lambda$ (the null string). Symbols from Σ are appended to x as they are imported into the skin membrane during the computation. The CPA need not import a symbol (from Σ_α) at every step.

Note that in the "maximal parallelism" operating mode, an unbounded number of symbols from Σ can enter the skin membrane in one step since several rules of type (3) in the definition of G may be applicable to an unbounded number of ab pairs in the skin membrane. If the input symbols (i.e., from Σ) that enter the membrane in the step are $\sigma_1, ..., \sigma_k$ (note that k is not fixed), then $\sigma_{i_1}...\sigma_{i_k}$ is the string appended to x, where $i_1, ..., i_k$ is some nondeterministically chosen permutation of $1, ..., k$. Actually, it can be shown [9] that, in fact, we can assume without loss of generality that $k \leq 1$ (i.e., at most one symbol enters the membrane in each step). A string $x = \sigma_1...\sigma_n \in \Sigma^*$ is accepted if G has a halting computation after importing symbols $\sigma_1, ..., \sigma_n$ from the environment. It follows from the result in [19] that every recursively enumerable language can be accepted by a CPA under the maximal parallelism semantics, and vice-versa.

In the rest of the paper, CPAs are assumed to operate in sequential mode, unless otherwise noted. It turns out that such a CPA is equivalent to a partially blind multicounter machine (PBCM), which is a one-way nondeterministic finite automaton augmented with blind counters [7]. At every step, each counter can be incremented/decremented by 1 or not changed, but it cannot be tested for zero. When there is an attempt to decrement a zero counter, the machine gets stuck and the computation is aborted. The machine does not have to read an input symbol at every step (i.e., it can have ϵ moves). An input string w is accepted if, when the machine is started in a distinguished initial state with all counters zero, it processes all the input symbols in w and eventually enters an accepting state with all the counters zero. Note that if in a computation, a zero counter gets stuck (because of an attempt to decrement it), the computation is aborted and the input is not accepted. It is well known that if "testing for zero" is allowed (i.e., the counters are not partially blind), such a machine can accept every recursively enumerable language, even when there are only two counters [14]. However, PBCM's are strictly weaker than TM's – its emptiness problem (Is the accepted language empty?) and, hence, also the membership problem are decidable. This follows from the decidability of the reachability problem for VAS [13].

Theorem 4. *Language L is accepted by a CPA if and only if it can be accepted by a PBCM.*

In what follows, we let CPA(n) (resp., CPA(linear)) denote the class of languages accepted by CPA in quasirealtime (resp., linear time), and PBCM(n) (resp., PBCM(linear)) denote the class of languages accepted by PBCM in quasirealtime (resp., linear time). We also write COUNTER(n) to denote $\{L \mid L$ is accepted by a multicounter machine in quasirealtime$\}$. The following theorem then follows from similar results for PBCM [7].

Theorem 5. *1. CPA has a decidable emptiness problem.*
2. *CPA is a proper subset of the family of recursive languages.*
3. *CPA does not contain the language $L = (\{a^n b^n \mid n \geq 0\})^*$ and is not closed under Kleene +.*

4. *CPA(n) is not closed under Kleene +.*
5. *CPA(n) is not closed under complementation.*
6. *CPA(n) is a proper subset of COUNTER(n).*
7. *CPA(n) is a proper subset of CPA(linear).*
8. *CPA(linear) - COUNTER(n) is not empty.*

Lemma 1. *(from [7]) For every k, there is a language that can be accepted accepted by a quasirealtime PBCM with $(k+1)$ counters but not by any quasirealtime PBCM with k counters.*

We can now show the following result:

Theorem 6. *For every r, there is an $s > r$ and a language L that can be accepted by a quasirealtime CPA with s membranes but not by any quasirealtime CPA with r membranes.*

A cooperative P system acceptor with dissolution rules and bounded membrane creation rules can be defined in the usual manner (as in a CPA). Clearly, a CPA is a special case of a cooperative P system acceptor. Hence, a PBCM can be simulated by a cooperative system. For the converse, the constructions in Theorems 1, 2, and 4 can be implemented on a PBCM. Hence, Theorems 4, 5, and 6 hold when CPA is replaced by PBCM. We have:

Corollary 1. *The following are equivalent: CPA, PBCM, cooperative P system acceptor with dissolution and bounded membrane creation rules.*

In Theorem 4, the model of the CPA G has a sequence of input symbols coming in from the environment, and this sequence constitutes the (one-way) input to the PBCM. Now suppose we modify the way the input is given to G. As before, let $\Sigma = \{a_1, ..., a_k\}$ and $\Sigma_\alpha = \Sigma \cup \{\alpha\}$, where α is a distinguished symbol. At the start of the computation, G is given a multiset $wa_1^{i_1}...a_k^{i_k}$ in its input membrane, where w is some fixed multiset from an alphabet Δ disjoint from Σ_α, and each $i_j \geq 0, 1 \leq j \leq k$. The environment only contains an abundance of α. Initially, there are no symbols from $\Sigma \cup \Delta$ in the environment, and only symbols from this set that are exported to the environment (during the computation) can be imported from the environment. Thus, the multiplicity of each symbol in $\Sigma \cup \Delta$ in the system (including the environment) remains the same during the computation. We say that $a_1^{i_1}...a_k^{i_k}$ is accepted if G, when given $wa_1^{i_1}...a_k^{i_k}$ in its input membrane, halts. We denote the language accepted by $L(G)$. We call the new model NCPA. Like CPAs, the language accepted by an NCPA can also be accepted by a PBCM. Hence, the following holds.

Theorem 7. *We can effectively construct, given an NCPA G, a PBCM M accepting $L(G)$. Hence, the emptiness problem for NCPAs is decidable.*

4 1-Deterministic P Systems with Prioritized Rules

Now let us look at P systems which allow rules to be prioritized, meaning that a rule of lower priority can only be used when rules of higher priority are no longer applicable.

Previously we showed that sequential cooperative P systems without priority rules are equivalent to VASS and hence nonuniversal. Allowing priority rules increases the power of these systems causing them to be universal. In fact, even cooperative P systems with the restriction of 1-determinism with only one membrane are already universal. We have the following result:

Theorem 8. *A prioritized 1-deterministic 1-membrane cooperative system (COS) with rules of the form $u \rightarrow v$, where $|u| = 2$, and $|v| = 1$ or 2, can simulate a 2-counter machine (hence, it is universal).*

The above result applies to a 1-deterministic 1-membrane symport/antiport system (SAS) [12, 16]. This system has rules of the following forms: $(x, out; y, in)$ which is an antiport rule, and (x, out) or (x, in) which is an symport rule, where x, y are strings of symbols. The *radius* of an antiport rule is $(|x|, |y|)$. For a symport rule, the radius is $|x|$.

Corollary 2. *A prioritized 1-deterministic 1-membrane symport/antiport system (SAS) whose rules are antiport of radius (2,1) or (2,2) can simulate a 2-counter machine.*

Looking at the class of communicating P systems, we find similar results. The class of 1-deterministic 1-membrane CPS with priority rules can also simulate a 2-counter machine using a technique similar to the proof of Theorem 8.

Theorem 9. *A prioritized 1-deterministic 1-membrane CPS can simulate a 2-counter machine.*

The above theorem is in contrast to a result in [11] that a sequential multi-membrane CPS whose rules are not prioritized is equivalent to a VAS.

Finally, consider the model of a sequential multi-membrane CS where the rules are prioritized. Specifically, there is a priority relation on the rules: A catalytic rule R' of lower priority than R cannot be applied if R is applicable. We refer to this system as prioritized CS. We know that the reachability set of a sequential multi-membrane CS is semilinear and, hence, its reachability problem is NP-complete. In [10], the status of the reachability problem for systems with prioritized rules was left open. Here we show that the reachability problem is also NP-complete. Taking advantage of the equivalence between sequential multi-membrane CS and communication-free VAS[1] [10], we first show the reachability problem for prioritized communication-free VAS (which will be defined in detail below) to be NP-complete, which immediately yields the mentioned complexity result for prioritized sequential multi-membrane CS. For related results concerning other types of prioritized concurrent models, the reader is referred to, e.g., [20].

Given a communication-free VAS $G = \langle x, W \rangle$, a *priority relation* ρ over W is an irreflexive, asymmetric, and transitive relation such that v_2 takes precedence over v_1 if $(v_1, v_2) \in \rho$, meaning that v_1 cannot be applied if v_2 is applicable. Due to the nature of communication-freeness, we further assume ρ to satisfy a property that if v and v' subtract from the same coordinate, neither (v, v') nor (v', v) is in ρ; otherwise, one of the two could never be applied. Let $\bar{\rho}$ denote $\{(v, v') | (v, v') \notin \rho \text{ and } (v', v) \notin \rho\}$. In this paper, $\bar{\rho}$ is assumed to be an equivalence relation.

[1] A communication-free VAS is a VAS where in every transition, at most one component is negative, and if negative, its value is -1

Theorem 10. *The reachability problem for prioritized communication-free VAS is NP-complete.*

We now have the following:

Corollary 3. *The reachability problem for sequential multi-membrane catalytic systems with prioritized rules is NP-complete.*

5 1-Deterministic Language Acceptors with Prioritized Rules

It is known that any recursively enumerable unary language $L \subseteq o^*$ can be accepted by a (deterministic) 2-counter machine M with a one-way input tape which operates as follows: M, starting in its start state with both counters zero, when given a string $o^n\$$, reads the input symbols left-to-right and halts if and only if $o^n \in L$. Note that M need not read a new input symbol at every step. We may assume that after the right delimiter is read by the counter machine, it can continue with the computation (without attempting to read another symbol on the tape). The input is accepted if M eventually halts. The input is not accepted if the machine does not halt.

The 2-counter machine can be normalized, with each instruction taking one of the following forms: (1) $\delta(q, \epsilon, positive, na) = (q', -1, d)$; (2) $\delta(q, a, na, na) = (q', 0, 0)$, where $a = o$ or \$ (meaning that q is a reading state and the machine reads an input symbol and changes state without altering the counter contents); (3) $\delta(p, \epsilon, na, positive) = (p', d, -1)$; (4) $\delta(p, a, na, na) = (p', 0, 0)$, where $a = o$ or \$; (5) $\delta(q, \epsilon, zero, positive) = (p, d, -1)$; (6) $\delta(p, \epsilon, positive, zero) = (q, -1, d)$. Note that the state can also be "tagged" whether they are in a reading mode or nonreading mode. Transitions of the form 1, 3, 5, 6 are non-reading (indicated by ϵ in the second parameter) and are handled as before. Transitions 2 and 4 are reading and are translated to cooperative rules $qa \rightarrow q'$ and $pa \rightarrow p'$, respectively.

It follows that all the results in the previous section can be reformulated for P systems that are acceptors when the 2-counter machine is an acceptor. For example, Theorem 9 can be written to read:

Corollary 4. *Every recursively enumerable unary language L can be accepted by a deterministic 1-membrane CPS with prioritized rules.*

6 Conclusion

In this paper we investigated the computational power of different types of sequential and 1-deterministic P systems. We showed that without prioritized rules, sequential P systems are equivalent to VAS, even if they are allowed to have dissolution rules and bounded creation rules. We also showed that these systems when used as language acceptors are equivalent to communicating P system acceptors which, in turn, are equivalent to to partially blind counter machines. When the rules are prioritized, there are two types of results: there are sequential P systems that are universal and sequential P systems that are nonuniversal. In particular, both communicating and cooperative

P systems are universal, even if restricted to 1-deterministic systems with one membrane. However, catalytic P systems with prioritized rules have NP-complete reachability problem and, hence, nonuniversal.

References

1. E. Csuhaj-Varju, O. Ibarra, and G. Vaszil. On the computational complexity of P automata. *Proc. DNA 10*, 2004.
2. Z. Dang and O. Ibarra. On P systems operating in sequential mode. *Pre-Proc. DCFS'04*, 2004.
3. R. Freund, Sequential P-systems, Available at *http://psystems.disco.unimib.it*, 2000.
4. R. Freund, Asynchronous P systems, in *Pre-Proc. Fifth Workshop on Membrane Computing*, eds. G. Mauri, Gh. Paun, and C. Zandron (2004).
5. R. Freund and Gh. Paun. On deterministic P systems. *See P Systems Web Page at http://psystems.disco.unimib.it*, 2003.
6. P. Frisco. About P systems with symport/antiport. *Second Brainstorming Week on Membrane Computing, Sevilla, Spain*, pp.224-236, Feb 2-7, 2004.
7. S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theor. Comput. Sci.* 7:311-324, 1978.
8. J. Hopcroft and J. J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8(2):135–159, 1979.
9. O. Ibarra. On the computational complexity of membrane systems. *Theor. Comput. Sci.* 320(1): 89-109, 2004.
10. O. Ibarra, Z. Dang, and O. Egecioglu. Catalytic P systems, semilinear sets, and vector addition systems. *Theor. Comput. Sci.*, 11(1):167–181, 2004.
11. O. Ibarra, H. Yen, and Z. Dang. On the power of maximal parallelism in P systems. *Proc. DLT*, (LNCS 3340), pp. 212-224, 2004.
12. C. Martin-Vide, A. Paun, and Gh. Paun. On the power of P systems with symport rules. *Journal of Universal Computer Science*, 8(2):317–331, 2002.
13. E. Mayr. An algorithm for the general Petri net reachability problem, *SIAM J. Computing*, 13(3):441–460, 1984.
14. M. Minsky. Recursive unsolvability of Post's problem of tag and other topics in the theory of Turing machines. *Ann. of Math*, 74:437-455, 1961.
15. M. Mutyam and K. Kriithivasan. P systems with active objects: Universality and efficiency. *Proc. of the 3rd Int'l Conf. on Machines, Computations, and Universality*, pp. 276-287, May 23-27, 2001.
16. A. Paun and Gh. Paun. The power of communication: P systems with symport/antiport, *New Generation Computing*, 20(3):295–306, 2002.
17. Gh. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.
18. Gh. Paun. Membrane Computing: An Introduction. *Springer-Verlag*, 2002.
19. P. Sosik. P systems versus register machines: Two universality proofs. In *Pre-Proceedings of Workshop on Membrane Computing (WMC-CdeA2002), Curtea de Arges, Romania*, pp. 371–382, 2002.
20. H. Yen. Priority conflict-free Petri nets. *Acta Informatica*, 35(8):673-688, 1998.

Global Optimality Conditions and Near-Perfect Optimization in Coding

Xiaofei Huang

School of Information Science and Technology
Tsinghua University, Beijing, P.R. China, 100084
huangxiaofei@ieee.org

Abstract. Finding ways of recognizing global optimum is the very fundamental, unsolved problem in existing optimization theories. We can not establish a true theory of optimization without it. Also, it is very hard to construct effective algorithms for finding global optimum. This paper presented a new optimization principle, called cooperative optimization, for solving this extremely important problem in optimization theory. A number of global optimality conditions are provided in a general form. The application of cooperative optimization in coding yields near-perfect results in finding global optima, significantly better than the most powerful optimization algorithm ever found so far.

1 Introduction

Optimization is a core problem both in mathematics and computer science [1]. However, there is no satisfying theory of global optimization up to now. The fundamental reason is the lack of a theory for recognizing global optimum. Without it, it is very hard to construct effective algorithms for finding global optimum. Except a few special cases, the existing theories can only identify local optimum. Often times, an optimization problem instance may have only one global optimum but many local optima and the number of them can be overwhelming. It defies the effectiveness of any search strategies built upon those theories.

To solve this very fundamental problem, a new principle of global optimization, called cooperative optimization, has been proposed [2] which is completely different any existing one. Its theoretical foundation has been established [3]. Within it there are a number of sufficient conditions for recognizing global optimum in a very general form [3]. Optimization algorithms of different computational powers are also derived based on this new principle [4]. A system based on those algorithms has a unique equilibrium. It converges to the equilibrium with an exponential rate regardless of initial conditions. In many important cases, it guarantees to find the global optima for difficult optimization problems when conventional methods often fail.

This paper presents the cooperative optimization principle in the language of mathematics in a more general form. This generalization leads to more powerful forms of cooperative optimization. The application of generalized cooperative optimization algorithms in coding is also given in this paper. It demonstrated

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 915–924, 2005.

an outstanding performance that significantly outperforms the most powerful optimization algorithm ever found so far. Optimization plays a key important role in coding. Coding is at the heart of data communications and data storage, two corner stones of modern information age. Many technologies we use daily have coding involved, such as music CDs, video DVDs, HDTVs, digital radios, cellular phones, hard disks and memories in our computers, modems that connect our computers to the Internet, and many more. A technology advance in coding can have a huge commercial value and a profound impact on everyone's life.

2 Global Optimization Problem

Many optimization problems can be formulated as finding the global minimum of an objective function of a general aggregate form

$$\min_{x_1,x_2,\ldots,x_n} \sum_{j=1}^{m} f_j(X_j) \quad \text{where } X_j \subset X \equiv \{x_1, x_2, \ldots, x_n\} \ . \tag{1}$$

The objective function in (1) is denoted as $E(x_1, x_2, \ldots, x_n)$ in this paper, or simply $E(x)$. It is also referred as the energy function or the cost function. $f_i(X_i)$ in (1) is a real valued function defined on a subset of variables X_i. It is also referred to as a constraint or a component function of E.

Variable x_i in (1) can be is a discrete variable or a continuous variable. If all variables are continuous and $f_j(X_j)$s are nonlinear, the optimization problem (1) is called nonlinear unconstrained optimization (Chapter 5 in [1]). If all variables are discrete, the problem is called combinatorial optimization (Chapter 3 in [1]).

The classical optimality condition for unconstrained nonlinear optimization problems is

$$\bigtriangledown E(x^*) = 0, \quad \text{where } \bigtriangledown \text{ is the gradient operator} \ . \tag{2}$$

This condition is at the basis of most classical algorithms, such as Armijo's line-search algorithm, gradient methods, conjugate gradient methods, Newton's method, Quasi-Newton methods, and derivative-free methods. Because this optimality condition is only a local optimality condition, those algorithms only guarantee to find a local optimal solution, not necessarily the global optimal one. Often times, to a problem instance, there is only one global optimum and many local optima. The local optimality does not guarantee the global optimality. To make the situation worse, those conventional optimization methods even do not know if the solutions they found are the global optimal ones or not.

When all variables are discrete, the problem in the form of (1) is a very general combinatorial optimization problem. The satisfiability problem, which is NP complete and a core problem in computer science, can be described in this form. The famous traveling salesman problem, which is NP hard, can also be formulated in this way.

To find global optimal solution of combinatorial optimization, there are exact methods like branch-and-bound, branch-and-cut, and dynamic programming.

Those methods are not efficient in producing solutions because they find solutions through exhaustive enumeration in one way or another. As the size of the instances increases, the computing time of these methods goes up quickly with a rate exponential to the size. Because of the complexity, people often employ heuristics. The popular ones are local search, swarm optimization, simulated annealing, tabu search, genetic algorithms, and variable neighborhood search. Those methods are based on empirical rules and have no guaranteed effectiveness. They also do not know when to stop searching because there is no solution for recognizing global optima.

3 A Simple Sufficient Optimality Condition

To find and recognize the global optimum of a complex objective function, we can use another function, called profile function, to help us. A profile function is simpler than the original one in terms of finding global optimum.

Proposition 1. *Let E be an objective function and $E^{'}$ be another function defined on the same set of variables as E. Assume that $x^*(E)$ be the global minimum of E. Assume further that $x^*(E^{'})$ be the global minimum of $E^{'}$ and it is unique. If*

$$E^{'}(x^*(E^{'})) = E(x^*(E)) \ , \tag{3}$$

then $x^*(E^{'})$ must be the global optimum of E, i.e., $x^*(E^{'}) = x^*(E)$.

In contrary to the classic necessary optimality condition (2), the optimality condition (3) is sufficient. Hence, if we can find a profile function such that its global optimum is unique and satisfies Eq. (3), then the global optimum of the original objective function can be found by searching the global optimum of the profile function.

However, there are still two remaining questions. First one is: what forms can we choose for profile function $E^{'}$ such that the search for $x^*(E^{'})$ (the global minimum of the profile function) is not hard in computation? Second one is: how do we know that $x^*(E) = x^*(E^{'})$ without the knowledge about the optimal value $x^*(E)$ most of the time?

Cooperative optimization presented in this paper provides answers to these two basic questions. It is a global optimization method solely based on the use of profile functions to search and recognize optimal solutions.

4 Cooperative Optimization

Cooperative optimization deploys an iterative process which computes a profile function at each iteration. Because the profile function $E^{'}$ is introduced by the cooperative optimization, it is free in choosing its form as long as we can easily find its global optimum. A simple form used originally in [2, 3] is

$$E^{'}(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} c_i(x_i) \ . \tag{4}$$

$c_i(x_i)$ is a function defined on x_i. Any profile function in this form is called the unary profile function because all its component functions are unary (defined on one variable). Obviously, to find the global minimum of the above function is very simple,

$$x_i^*(E^{'}) = \arg\min_{x_i} c_i(x_i) \quad \text{for } i = 1, 2, \ldots, n \ .$$

The concept of profile function can help us to generalize the unary profile function further into any profile function, containing k-ary component functions ($k \geq 1$). One special form of generalization for the profile function (4) is to break $c_i(x_i)$ into several pieces,

$$c_{i1}(x_i), c_{i2}(x_i), \ldots, c_{iNi}(x_i) \ .$$

That is, $c_i(x_i) = \sum_{j=1}^{N_i} c_{ij}(x_i)$. Following that, the generalized profile function comes as

$$E^{'}(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} \sum_{j=1}^{N_i} c_{ij}(x_i) \ . \tag{5}$$

Such a simple generalization can greatly improve the performance of cooperative optimization. In the following discussions, all the equations and the theorems are given only for the original profile function (4). They can be generalized for the new profile function (5) in a straightforward way.

The cooperative optimization method computes a unary profile function at each iteration. This unary profile function is a lower bound function of the original objective function, guaranteed to be tightened after each iteration.

Cooperative optimization has a number of sufficient conditions for recognizing global optimum. Those conditions are all derived from the simple sufficient optimality condition (3). All of them can be used to identify global optimal solutions without the knowledge about the optimal value $x^*(E)$.

What is the basic idea behind cooperative optimization? How does it compute the profile function? We will introduce these in the following subsection using a simple example.

4.1 Basic Ideas

We can use a multi-agent system to solve a hard problem following the divide-and-conquer principle. We first break up the problem into a number of sub-problems of manageable sizes and complexities. Following that, we assign each sub-problem to an agent, and ask those agents to solve the sub-problems in a cooperative way. The cooperation is achieved by asking each agent to compromise its solution with the solutions of others instead of solving the sub-problems independently. We can make an analogy with team playing, where the team members work together to achieve the best for the team, but not necessarily the best for each member. In many cases, cooperation of this kind can dramatically improve the problem-solving capabilities of the agents as a team, even when each agent may have very limited power.

4.2 A Simple Example

Let the problem be the optimization of the following objective function

$$E(x_1, x_2, x_3) = f_{12}(x_1, x_2) + f_{23}(x_2, x_3) + f_{13}(x_1, x_3) \ , \tag{6}$$

which is expressed as an aggregation of three binary component functions, $f_{12}(x_1, x_2)$, $f_{23}(x_2, x_3)$, and $f_{13}(x_1, x_3)$. These functions are binary because each defines on two variables.

To illustrate the decomposition of this problem into simple sub-problems, we map the objective function (6) into a graph (shown in the upper portion of Fig. 1). We can view each variable as a node in the graph and each binary component function as a connection between two nodes. This graph has one loop and we can decompose it into three sub-graphs of no loop shown in the lower portion of Fig. 1, one for each variable (double circled). Each sub-graph is associated with one objective function, $E_i, i = 1, 2, 3$. Those objective functions are

$$E_1(x_1, x_2, x_3) = (f_{12}(x_1, x_2) + f_{13}(x_1, x_2))/2 \ .$$
$$E_2(x_1, x_2, x_3) = (f_{23}(x_2, x_3) + f_{12}(x_1, x_2))/2 \ ,$$
$$E_3(x_1, x_2, x_3) = (f_{13}(x_1, x_3) + f_{23}(x_2, x_3))/2 \ .$$

Obviously,

$$E = E_1 + E_2 + E_3 \ .$$

With such a decomposition, the original problem, $\min E$, becomes three sub-problems, $\min E_i$, for $i = 1, 2, 3$.

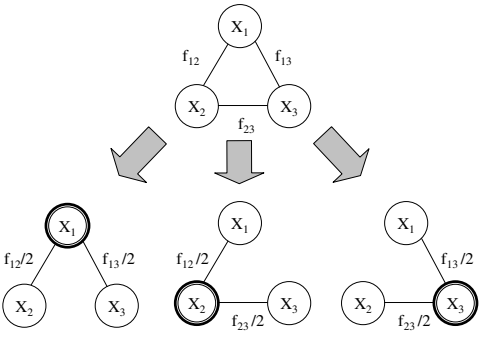


Fig. 1. The illustration of decomposing a graph into sub-graphs of tree-like structures

For the ith sub-problem, the preferences for picking values for variable x_i are used as the soft decisions for solving the sub-problem. Those preferences are measured by some real values and are described as a function of x_i, denoted as $c_i(x_i)$. It is also called the assignment constraint for variable x_i. The different

function values, $c_i(x_i)$, stand for the different preferences in picking values for variable x_i. Because we are dealing with minimizing E, for the convenience of the mathematical manipulation, we choose to use smaller function values, $c_i(x_i)$s, for more preferable variable values.

To introduce cooperation in solving the sub-problems, we iteratively update the assignment constraints (soft decisions in assigning variables) as

$$c_1^{(k)}(x_1) = \min_{x_2,x_3} (1-\lambda_k)\, E_1 + \lambda_k \sum_j (1/3) c_j^{(k-1)}(x_j) \tag{7}$$

$$c_2^{(k)}(x_2) = \min_{x_1,x_3} (1-\lambda_k)\, E_2 + \lambda_k \sum_j (1/3) c_j^{(k-1)}(x_j) \tag{8}$$

$$c_3^{(k)}(x_3) = \min_{x_1,x_2} (1-\lambda_k)\, E_3 + \lambda_k \sum_j (1/3) c_j^{(k-1)}(x_j) \tag{9}$$

where k is the iteration step.

It turns out that $c_1^{(k)}(x_1)$, $c_2^{(k)}(x_3)$, and $c_3^{(k)}(x_3)$ computed by the above different equations defines a lower bound function of E, i.e.,

$$c_1^{(k)}(x_1) + c_2^{(k)}(x_2) + c_3^{(k)}(x_3) \le E(x_1, x_2, x_3), \quad \text{for any } k\ .$$

Let

$$E_-^{(k)}(x_1, x_2, x_3) \equiv c_1^{(k)}(x_1) + c_2^{(k)}(x_2) + c_3^{(k)}(x_3)\ .$$

Assume that λ_k is fixed. Then it also turns out that the series $\{\min E_-^{(k)}\}$ is monotonically non-decreasing, i.e.,

$$\min E_-^{(1)} \le \min E_-^{(2)} \le \ldots \le \min E_-^{(k)}.$$

Hence, $E_-^{(k)}$ is the profile function E' computed by cooperative optimization at iteration k. We use the notation $E_-^{(k)}$ to denote that it is a lower bound based profile function.

Assume that

$$\lim_{k\to\infty} \min E_-^{(k)} = E^*\ ,$$

then the global optimum is found.

Variable x_1 is contained in the three sub-problems. At each iteration, x_1 has a value in the optimal solution for each of the sub-problems. Those values may not be the same. If all of them are of the same value, we say that all the sub-problems reach a consensus assignment for variable x_1. If all sub-problems reach a consensus assignment for each variable (x_1, x_2, and x_3), we say that a consensus solution is reached at the iteration. Consensus solution is an important concept of cooperative optimization for defining global optimality conditions. Normally, a consensus solution is the global optimum.

Parameter λ_k controls the level of the cooperation at iteration step k. It is so called the cooperation strength, satisfying $0 \le \lambda_k < 1$. A higher value for λ_k will weigh the solutions of the other sub-problems $c_j(x_j)$ more than the one of the

current sub-problem E_i. In other words, the solution of each sub-problem will compromise more with the solutions of other sub-problems. As a consequence, the cooperation in optimizing the sub-problems is stronger. The system is more likely to find a consensus solution in solving the sub-problems, i.e., finding the global optimum.

The update functions, (7),(8), and (9), are a set of difference equations of the assignment constraints $c_i(x_i)$. Unlike conventional difference equations used by probabilistic relaxation algorithms [5], and Hopfield Networks [6], this set of difference equations always has one and only one equilibrium given λ. Some important properties of this cooperative optimization will be shown in the following subsections.

4.3 Cooperative Optimization in a General Form

Let $E(x_1, x_2, \ldots, x_n)$ be a multivariate cost function, or simply denoted as $E(x)$, where each variable x_i has a finite domain D_i of size m_i ($m_i = |D_i|$). We break the function into n sub-functions E_i ($i = 1, 2, \ldots, n$), one for each variable, such that E_i contains at least variable x_i, the minimization of each sub-function E_i (the sub-problem) is computational manageable in complexity, and

$$E(x) = \sum_{i=1}^{n} E_i(x). \tag{10}$$

The cooperative optimization is defined by the following set of difference equations:

$$c_i^{(k)}(x_i) = \min_{x_j \in X_i \setminus x_i} \left((1 - \lambda_k)\, E_i + \lambda_k \sum_j w_{ij} c_j^{(k-1)}(x_j) \right) . \tag{11}$$

λ_k ($k = 1, 2, \ldots$) and w_{ij} ($1 \le i, j \le n$) are parameters we need to choose. As mentioned before, parameter λ_k controls the level of the cooperation at step k. Stronger cooperation levels may lead to better solutions, but slower convergence rates. Intuitively, we might choose w_{ij} such that it is non-zero if x_j is contained by E_i. That asks the optimization of E_i to take into consideration of the variable assignment decision, $c_j^{(k-1)}(x_j)$, made by the optimization of E_j at a previous step. However, theory tells us that this is too restrictive. To make the algorithm work, we need to choose $(w_{ij})_{n \times n}$ to be a propagation matrix defined as follows:

Definition 1. *A propagation matrix $W = (w_{ij})_{n \times n}$ is a irreducible, nonnegative, real-valued square matrix and satisfies*

$$\sum_{i=1}^{n} w_{ij} = 1, \quad \textit{for } 1 \le j \le n \ .$$

Definition 2. *The system is called reaching a consensus solution at step k if, for any i and j where E_j contains x_i,*

$$\arg\min_{x_i} \tilde{E}_i^{(k)} = \arg\min_{x_i} \tilde{E}_j^{(k)} \ ,$$

where $\tilde{E}_i^{(k)}$ is defined as $\tilde{E}_i^{(k)} \equiv (1-\lambda_k)\, E_i + \lambda_k \sum_j w_{ij} c_j^{(k-1)}(x_j)$.

To simplify the notations in the following discussions, let $c^{(k)} \equiv (c_1^{(k)}, \ldots, c_n^{(k)})$. Let $\tilde{x}_i^{(k)} \equiv \arg\min_{x_i} c_i^{(k)}(x_i)$, the favorable value for assigning variable x_i. Let $\tilde{x}^{(k)} \equiv (\tilde{x}_1^{(k)}, \ldots, \tilde{x}_n^{(k)})$. It is the candidate solution obtained at iteration k.

4.4 Global Optimality Conditions

To recognize global optimum, there are sufficient conditions and necessary conditions. Sufficient conditions are used to identify if a solution is the global optimum or not. Necessary conditions are used to determine if any variable value can be in the global optimum. They allow us to eliminate variable values that can not be in any global optimum. If only one value is left for each variable after the value elimination, the global optimum is found which consists of the remaining values. All these conditions are all referred to as global optimality conditions to differentiate them in name from the classical local optimality conditions.

Theorem 1 (Sufficient Condition 1). *If a consensus, $\tilde{x}$, is found at some step with the choice of $\lambda = 0$, then the consensus is also a global optimum.*

This sufficient condition is, therefore, a weak sufficient condition since the possibility of finding a consensus without cooperation ($\lambda = 0$) is quite low in dealing with complex problems.

Theorem 2 (Sufficient Condition 2). *Given a propagation matrix W and the general initial condition $c^{(0)} = 0$ or $\lambda_1 = 0$. If $E_-^{(k+1)} \leq E_-^{(k)}$ at some step k, then a consensus solution found at that step is also a global optimum.*

The above theorem provides us the second sufficient condition for recognizing a global optimum. This sufficient condition does not restrict the choice of cooperation strength λ. The whole range of the cooperation strength can be explored to increase the chance of finding a consensus solution.

The second sufficient condition is stronger than the first one. Given any problem, if a global optimum can be found under the first sufficient condition, it can also be found under the second sufficient condition. At the same time, there exist some problem instances whose global optima can be found under the second sufficient condition only. Intuitively, the possibility of finding the consensus solution is much higher for the cooperative system with cooperation ($\lambda > 0$) than the one without cooperation ($\lambda = 0$).

Theorem 3 (Sufficient Condition 3). *Given the propagation matrix* $W = (1/n)_{n\times n}$*, and the general initial condition* $c^{(0)} = 0$ *or* $\lambda_1 = 0$*. If a consensus* $\tilde{x}$ *is found at each iteration from step* k_1 *to step* k_2 *with* λ *having a fixed value, and the second minimum value of the variable assignment constraint* $c^{(k_2)}(\hat{x}_i)$ *satisfies the following inequality:*

$$c^{(k_2)}(\hat{x}_i) > \lambda^{(k_2-k_1)}(E(\tilde{x}) - E_-^{(k_1)}) + \lambda E(\tilde{x})/n + (1-\lambda)E_i(\tilde{x}), \tag{12}$$

for all i*, then* $\tilde{x}$ *is a global optimum.*

This sufficient condition does not restrict the choice of cooperation strength λ. The whole range of the cooperation strength can be explored to increase the chance of finding a consensus.

Theorem 4 (Necessary Condition). *Given a propagation matrix* W*, and the general initial condition* $c^{(0)} = 0$ *or* $\lambda_1 = 0$*. If value* x_i^* $(x_i^* \in D_i)$ *is in the global optimum, then* $c_i^{(k)}(x_i^*)$*, for any* $k \geq 1$*, must satisfy the following inequality,*

$$c_i^{(k)}(x_i^*) \leq (E^* - E_-^{*(k)}) + c_i^{(k)}(\tilde{x}_i^{(k)}) \tag{13}$$

where $E_-^{*(k)}$ *is, as defined before, a lower bound on* E^* *obtained by the cooperative system at step* k*.*

In practice, we use an upper bound E^+ instead of the optimal value E^* in (13) because the former is easier to obtain than the latter.

5 Application in Coding

Recently, people found that turbo codes [7] and LDPC codes [8] can approach the Shannon limit for channel coding. They will be the key to the next generation of communications. Both types of codes use the most powerful optimization algorithm, called the sum-product algorithm [9], ever found so far for decoding. The popular optimization methods, such as simulated annealing and generic algorithms, have terrible performances in solving the combinatorial problems raised from decoding. To the surprise of many mathematicians, we have little theoretical understanding of its computational properties despite of its effectiveness.

Two LDPC codes are used in our experiments. The first one defines a combinatorial optimization problem with 32,768 variables and 15,961 component functions. The data rate is 0.513. 500 instances are randomly generated. If all global optima for these 500 instances are found, the error rate is zero. The second one has 46,656 variables and 31,031 component functions. The data rate is 0.335 and 100 instances are randomly generated. The performance comparison is shown in Figure 2. In decoding both codes, the cooperative optimization algorithm has much less bit error rates (BER) than those of the sum-product algorithm. As the signal/noise rate (Eb/No) in decibel increases, the BER rates of the cooperative optimization algorithm quickly drop to zero, much faster than those of the sum-product algorithm.

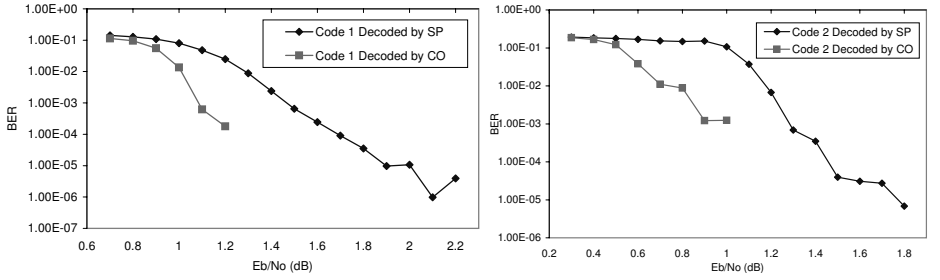


Fig. 2. Performance comparison of the sum-product algorithm (SP) and the cooperative optimization algorithm (CO) in decoding two LPDC codes

6 Conclusions

Cooperative optimization defines a basic principle for global optimization. It solved a fundamental problem in existing optimization theories by offering global optimality conditions. Based on that principle, cooperative optimization algorithms are constructed which can be more efficient and effective than conventional ones. The cooperative optimization algorithms have many excellent computational properties, most of them are not possessed by any classical ones.

References

1. Pardalos, P., Resende, M.: Handbook of Applied Optimization. Oxford University Press, Inc. (2002)
2. Huang, X.: A polynomial-time algorithm for solving np-hard problems in practice. SIGACT Newsletter **34** (2003) 101–108
3. Huang, X.: Cooperative optimization for solving large scale combinatorial problems. In: Theory and Algorithms for Cooperative Systems. Series on Computers and Operations Research. World Scientific (2004) 117–156
4. Huang, X.: A general framework for constructing cooperative global optimization algorithms. 4th International Conference on Frontiers in Global Optimization (2003)
5. Rosenfeld, A., Hummel, R., Zucker, S.: Scene labelling by relaxation operations. IEEE Transactions on System, Man, and Cybernetics **SMC-6** (1976) 420
6. Hopfield, J.: Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences **79** (1982) 2554–2558
7. Berrou, C., Glavieux, A., Thitimajshima, P.: Near shannon limit error-correcting coding and decoding: turbo codes. In: Proceedings of the 1993 IEEE International Conference on Communication. (1993) 1064–1070
8. Gallager, R.G.: Low-Density Parity-Check Codes. PhD thesis, Department of Electrical Engineering, M.I.T., Cambridge, Mass. (1963)
9. Kschischang, F.R., Frey, B.J., andrea Loeliger, H.: Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory **47** (2001) 498–519

Angel, Devil, and King*

Martin Kutz[1,**] and Attila Pór[2]

[1] Max-Planck-Institut für Informatik, Saarbrücken, Germany
mkutz@mpi-sb.mpg.de
[2] CASE Western Reserve University, Cleveland, USA
apor@renyi.hu

Abstract. The Angel-Devil game is played on an infinite chess board. In each turn *the Angel* jumps from his current position to a square at distance at most k. He tries to escape his opponent, *the Devil*, who blocks one square in each move. It is an open question whether an Angel of some power k can escape forever. We consider Kings, who are Angels that can only walk, not jump. Introducing a general notion of speed for such modified pieces, we obtain an improvement on the current best Devil strategy. Our result, based on a recursive construction of dynamic fractal barriers, allows the Devil to encircle Kings of any speed below 2.

1 Introduction

Two players, *the Angel* and *the Devil*, play a game on an infinite chess board whose squares be indexed by pairs of integers. The Angel is an actual "person" moving across the board like some chess piece, while his opponent does not live on the board but only manipulates it. In each move, the Devil blocks an arbitrary square of the board such that this location may no longer be stepped upon by the Angel. The Angel in turn, flies in each move from his current position $(x, y) \in \mathbb{Z}^2$ to some unblocked square at distance at most k for some fixed integer k, i.e., to some position $(x', y') \neq (x, y)$ with $|x'-x|, |y'-y| \leq k$. Note that Devil moves are not restricted to the Angel's proximity or limited by any other distance bounds; he can pick squares at completely arbitrary locations.

The Devil wins if he can stop the Angel, that is, if he manages to get him in a position with all squares in the $(2k+1) \times (2k+1)$ area around him blocked. The Angel wins if he succeeds to fly on forever. The open question is, whether for some sufficiently large integer k the Angel with distance bound k, called the *k-Angel*, can win this game.

First variants of this game were discussed by Martin Gardner [7], who names D. Silverman and R. Epstein as original inventors. Though his article deals mainly with finite configurations, i.e., the question whether a chess king (which is simply a 1-Angel) can reach the boundary of a given rectangular board, he also asks for a strategy against a chess knight on an infinite board. In its present

* This work was previously published as part of the first author's PhD thesis [9]

** Supported by the Deutsche Forschungsgemeinschaft within the European graduate program "Combinatorics, Geometry, and Computation" (GRK 588/2)

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 925–934, 2005.

form the Angel game first appeared in Berlekamp, Conway, and Guy's classic [2] (Chapter 19). Amongst detailed analyses of games with kings and other chess pieces on finite boards against Devils with certain additional restrictions, the authors coin the names "Angel" and "Devil" and give a thorough proof that the chess king can be caught. Then Conway [4] focused entirely on the infinite Angel game, trying to explain possible pitfalls with certain natural escape attempts and pointing out the hardness of the problem. Besides all variants, the central open question remains whether some Angel of sufficient power can escape forever. In his overview article [5], Demaine cites it as a difficult unsolved problem of combinatorial game theory.

In this work, we present an improvement on the current best known Devil strategy. Therefore we introduce a reformulation of the original game, which allows us to focus on speed as the important parameter. We define a k-King to be a k-Angel who cannot fly but runs; that is, a k-King is allowed to make k ordinary chess-king steps per turn, where each single step has to use an unblocked square. We shall see that Kings and Angels are asymptotically equivalent (if some Angel can escape then also some King can, and vice versa) and that the concept of k-Kings naturally extends to fractional and even irrational speed (Definition 1). Our main result is the following:

Theorem 1. *The Devil can catch any α-King with $\alpha < 2$.*

Many proof details, which have to be omitted here due to space constrains, can be found in the first author's thesis [9].

2 Basic Facts and Previous Results

The only case for which the k-Angel problem is solved is $k = 1$, the ordinary chess king. We like to sketch a winning strategy for the Devil, which resembles the analysis in [2]. These key ideas form the starting point for our proof of Theorem 1.

Assume the Devil wants to prevent the king from crossing a certain horizontal line. With three squares above the king already blocked on that line, like in the left of Figure 1, this is easily achieved. The Devil simply answers a king move a to the right with an extension of that triple block by a play at u. A further move to b is countered by v and likewise, a left movement to a' is blocked at u'. Pushing along in this simple fashion ensures that the king cannot cross. It is not difficult to get the three initial blocks placed on a blank line when a king is just approaching. By inspection of cases, one can show that five approach moves suffice for the Devil to create the basic triple.

The right of Figure 1 indicates how to turn the pushing argument into a Devil win. With his first moves, the Devil blocks a finite number of squares in the four corners of an imaginary box around the king, which is chosen large enough to ensure that during this preparatory phase the king does not get too close to the boundary. After that, the Devil plays the above wall-pushing strategy along the dotted lines whenever the king approaches the border. The solid corners are

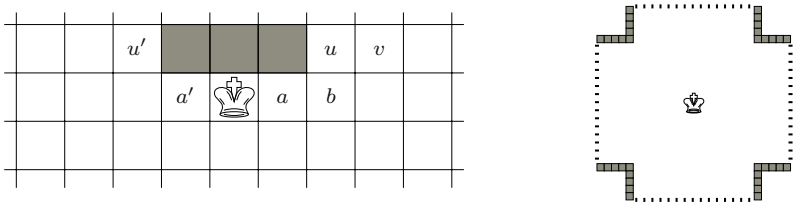


Fig. 1. Pushing the chess king along a line (left) and catching him in a box (right)

there to ensure that the Devil is never forced to play on two fronts at the same time.

The Fool argument. The first general idea for an escape with a k-Angel might be to run away in one direction. With sufficiently high power k, should not the Angel be simply too fast for the Devil? The answer is *no*. Conway [4] defines a *k-Fool* to be a k-Angel who commits himself to strictly increasing his y-coordinate in every move. He shows that a Fool of any power k can be caught. The Fool counter already indicates that devising an escape strategy for some Angel might be a very difficult task. By a dove-tailing argument the result can even be turned into the following surprising fact [4].

Theorem 2 ("Blass-Conway diverting strategy"). *There is a strategy for the Devil with the following property. For each point p of the plane and each distance d, no matter how the Angel moves, there will be two times $t_1 < t_2$ such that at time t_2 the Angel will be d units nearer to p than at time t_1.*

Angels in higher dimensions. For three dimensions, the Angel problem is solved. Independently, the first author [9] and Bollobás and Leader [3] proved that in $\mathbb{Z}^3$, and thus also in all higher dimensions, some Angel can escape.

Winning and losing infinite games. General infinite games may behave a little peculiar in so far as a clear winner need not always exist. The axiom of choice allows the construction of games in which neither player has a winning strategy, even though the game does not allow for draws [8, Sec. 43]. However, it is known [6, 10] that for reasonably well-behaved games this cannot happen: they are *determined*. For the Angel-Devil game it is not hard to show this property directly by a compactness argument, so we know that either the Angel or the Devil must have a winning strategy.

A further useful observation is that in a sense, the game is infinite only from the point of the Angel. If the Devil wins, the game ends, by definition, after finitely many moves. An application of König's lemma shows that in this case the Angel cannot delay his defeat arbitrarily.

Lemma 1. *If the Devil has a winning strategy against some Angel, then there exists a bound N such that the Devil can stop that Angel in at most N moves. Conversely, if the Angel can survive for any arbitrarily large, previously given number of steps then he can escape forever.*

3 The Need for Speed

There is pretty little known about even very weak Angels. Already the destiny of the 2-Angel is not settled and even more, it is unknown whether a chess knight can be caught. We do not have a solution for the 2-Angel, either, but we make a first step in this direction by devising Devil strategies against opponents whose power lies somewhere between that of a 1-Angel and a 2-Angel. The improvement appears rather modest but the new concepts we need to introduce in order to obtain them or even state them, reveal details of the game that seem to lie hidden with Conway's original Angel.

Let us take a closer look at what happens when we upgrade the original chess king to a 2-Angel. This is already a large step; the improvement is actually two-fold. Not only does the 2-Angel move at twice the speed, any barriers must also be twice as thick to hold him back. In a sense, the 2-Angel can be said to be 4 times stronger than the 1-Angel. We focus on the first aspect: *speed.* The Angel's ability to jump over obstacles shall be suppressed as an undesired side effect. Define a *k-King* as a player who in each turn makes exactly k ordinary chess-king moves, while the Devil still gets to place one block per turn. The point is that every single chess-king move must be valid. The k-King cannot fly.

If we want to use Kings for the study of the Angel problem, they should, in some qualitative sense at least, be equivalent to Angels. Obviously, a k-Angel is stronger than a k-King: An escape strategy for a King can be used for an Angel of the same power as well. The converse is, of course, not true—not for trivial reasons at least—but we can show that if you can catch Kings of arbitrary power k then you can also catch any Angel. Of course, the reduction from Angels to Kings requires an increase in speed.

Proposition 1. *If the k-Angel can escape then so can the $99k^2$-King.*

Proof (sketch). We derive an escape strategy for the $99k^2$-King from an escape strategy for the k-Angel. While the King plays against the "real" Devil, we set up an additional, imaginary board with an imaginary k-Angel, where we simulate the action on the King's board through appropriate transformations. The King's board is partitioned into a regular grid of sidelength-$18k^2$ boxes. Likewise, the Angel's board is segmented into blocks of sidelength k. The boxes of the two worlds are in one-to-one correspondence with each other, in the obvious fashion: the starting points lie in corresponding boxes and further, all adjacencies are preserved. These partitions and the correspondences are fixed once and for all.

We play as follows. When the Devil blocks some square in the King's world, we cross out an arbitrary empty square from the corresponding box in the Angel's world or from one of the eight adjacent boxes there. When it is the King's turn, we use our escape strategy for the Angel to get a move in the imaginary world. This move is then translated into the King's plane by a movement of the King into the corresponding box there. If, for example, the Angel jumps from his current box into the next box to the north, then the King runs into the northern box in his world, too. The precise position within that box is completely independent of the Angel's position in his box, however. It depends on technical

details which we must skip here for brevity. They have to guarantee that the King only stops at locations from where the four lines into the four axis parallel directions within the current box are completely free. This invariant then ensures a free passage for the King into the next target box, which takes no more than $99k^2$ steps. □

We emphasize again that the quantitative proportion of the above reduction is not our main concern. The purpose of Proposition 1 is to establish the qualitative equivalence between Angels and Kings, as a legitimation to use Kings as a tool to attack the Angel problem.

4 Real Kings

For Theorem 1 to make sense at all, we need to define what Kings of fractional speed shall be. So what is a 3/2-King? On average he should get to make three King steps for 2 Devil steps, which we could realize by a move sequence like $KKKDDKKKDD\ldots$, which shall mean that the King makes 3 steps, then the Devil blocks 2 squares, and so on. However, such a concept would depend on the actual representation of a rational number. The 6/4-King would get a different sequence. We could get around this by demanding reduced fractions but then a 1001/8-King would behave completely different from a 1000/8-King, who should simply be the 125-King. What is worse, the grouping of Devil moves could be lethal for the King. For example, the eight consecutive Devil moves in the sequence $K^{1001}D^8K^{1001}D^8\ldots$ could be used to encircle the King completely, even though his average speed would be greater than 125.

What we want are move sequences that approximate a given speed $\alpha \in \mathbb{R}^+$ as fair as possible, avoiding unnecessarily large chunks of moves for either side. The sequence $(u_n)_{n\in\mathbb{N}}$ defined by

$$u_n = \lfloor (n+1)\gamma + \phi \rfloor - \lfloor n\gamma + \phi \rfloor \in \{0,1\} \quad \text{with } \gamma = \frac{\alpha}{\alpha+1} \in (0,1) \tag{1}$$

and some constant offset $\phi \in \mathbb{R}$ shows this behavior—if we interpret 1's in the sequence as King and 0's as Devil moves. The basic behavior of such *sturmian sequences* is easy to understand (see [1] for a broad treatment and for historic references). Expression (1) simply compares consecutive elements of the arithmetic progression $(n\gamma + \phi)$. Whenever there lies an integer between the nth and the $(n+1)$st element of $(n\gamma + \phi)$ we have $u_n = 1$, otherwise, when the two elements fall in a common integer gap, (1) evaluates to $u_n = 0$. We conclude that the frequency of 1's in (u_n) is γ; hence the frequency of 0's is $1-\gamma$ and we get (cf. [1])

$$\lim_{n\to\infty} \frac{|\{i \le n : u_i = 1\}|}{|\{i \le n : u_i = 0\}|} = \frac{\gamma}{1-\gamma} = \alpha.$$

Definition 1. *For $\alpha \in \mathbb{R}^+$ we define the α-*King *to be a King whose move sequence is given by (1) with $\phi = 0$. This means that in the n*th *time step the King moves by one square if $u_n = 1$ and the Devil gets to block a new square if $u_n = 0$.*

The choice of the offset ϕ looks arbitrary. For a natural definition it is desirable that the chances of the α-King in the game do not depend on this parameter. And in fact, this can be shown.

Lemma 2. *Any two Kings with move sequences generated by* (1) *with the same speed parameter* α *but different* ϕ*'s either can both escape or can both be caught.*

For integral α, the above definition of an α-King obviously coincides with the previous definition of a k-King. For $\alpha = k \in \mathbb{N}^+$, the defining sequence (1) produces exactly k many 1's between any two consecutive 0's, just as expected. It is also clear that our notion of an α-King fulfills our wish for fairness. Large chunks of Devil moves cannot occur. One easily checks that for $\alpha \geq 1$ the Devil never gets to block two squares at a time. On the other hand, we can guarantee that not only in the long run but also locally, the Devil always gets his share of moves.

Definition 2. *A 0/1-sequence is* (s,t)-bounded, $s,t \in \mathbb{N}^+$, *if every contiguous subword that contains strictly more than* s *occurrences of 1's contains at least* t *occurrences of 0's. We call a King with a given move sequence* (s,t)-bounded *if that sequence is* (s,t)*-bounded.*

Lemma 3. *An* α*-King,* $\alpha \in \mathbb{R}^+$, *is* (s,t)*-bounded for every pair* $s,t \in \mathbb{N}^+$ *with* $\alpha \leq s/t$.

The "strictly" in the definition appears for a technical reason. Namely, starting from any 1 in the sequence, we count 0's until we reach the $(s+1)$st 1. By then we have passed at least t many 0's. When we read on until the $(2s+1)$st 1 shows up, we are sure to have counted at least $2t$ many 0's. And so on. Before the $(rs+1)$st 1 appears, we are guaranteed to read at least rt many 0's.

5 Low-Density Barriers

Let us have a closer look at the Devil strategy against the 1-King from the beginning. It seems we wasted some potential there. After the preparation of the corners, the Devil simply sits and waits for the King to arrive at one of the four sides. We can exploit this potential advantage.

The basic idea for the King counter was our dynamic-wall argument, where we had the King pushing along a line without ever letting him break through. Against a 2-King the Devil would need some blocks already in place in order to carry out the same principle. With every second square blocked in advance, the King cannot break through. Starting from the initial position in Figure 2 with only two additional squares blocked, the Devil can push along with the 2-King by answering the double move a_1, a_2 at u, then b_1, b_2 at v, and so on.

How long would it take the Devil to prepare such a density-1/2 wall against the 2-King? Since he needs to block 1 square out of 2, he can set up such a wall at an absolute speed of 2, which is exactly the speed of the 2-King. In other words, the Devil can build such fences against the 2-King at the same speed the 2-King runs. However, for Theorem 1 this will not be enough, yet; we need barriers of lower, fractional densities.

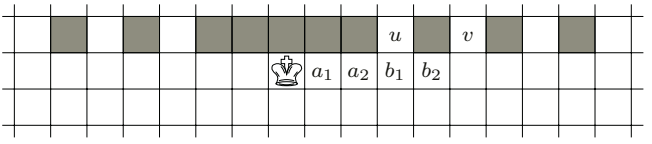


Fig. 2. A wall against the 2-King

Definition 3. *An* infinite (s,t)-fence *is an infinite horizontal or vertical strip in the plane with some squares blocked such that when an (s,t)-bounded King enters the strip from one side, the Devil can play in a way that prevents the King from leaving it on the other side. Formally, such a fence is a map $F\colon \mathbb{Z} \times [1\,..\,w] \to \{0,1\}$, where $F^{-1}(1)$ is the set of blocked squares. The integer w is called the* width *of F.*

We call such a fence periodic *if there exists some integer λ such that $F(x,y) = F(x+\lambda,y)$ for all $x \in \mathbb{Z}$. Call the minimum such λ the* period *of F. In this case we also define the* density *of the fence, as the ratio*

$$\frac{1}{\lambda}\left|\left\{(x,y) \mid 1 \le x \le \lambda, 1 \le y \le w, F(x,y) = 1\right\}\right|.$$

Note that density is measured with respect to length, not area. Width is not the crucial quantity, it appears for merely technical reasons.

Lemma 4. *Against an (s,t)-bounded King, $1 < s/t \le 2$, there exists a periodic infinite fence of density $1 - t/s$ and width $10s+1$.*

Proof (sketch). We follow the idea of Figure 2 for the 2-King. Define $F\colon \mathbb{Z} \times [1\,..\,10s+1] \to \{0,1\}$ by letting F be everywhere zero except at those points (x,y) with $0 \le x \bmod s < s-t$ and $y = 5s+1$. In other words, we group the central horizontal line $y = 5s+1$ into segments of s squares and place $s-t$ blocks in each segment, as shown in Figure 3. The density of this pattern is obviously the claimed $(s-t)/s$. We omit the precise mechanism of the fence due to space constraints. The width of $10s+1$ is required to grant the Devil some preparatory moves when the King enters the strip. □

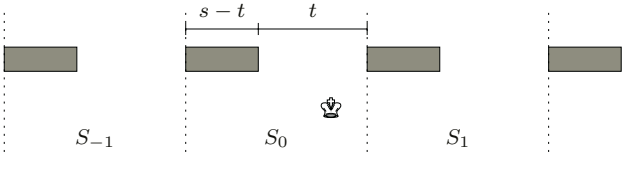


Fig. 3. An infinite (s,t)-fence

It is important to note that our fences are dynamic, in the sense that the Devil has to play in them while the King tries to break through. So the Devil will not have time to play somewhere else while he defends such a barrier. On the other hand, the density describes the construction cost which has to be

spent before the King reaches the fence. So what the Devil wants are fences of low density. Of course, he cannot build infinite structures in finite time. Infinite fences serve as a mere theoretical concept, which is easier to handle than finite fences, whose existence can be easily derived from the infinite ones.

Definition 4. *A* finite (s,t)-fence *is a rectangular box of size $\ell \times w$ in the plane with some squares blocked, such that when an (s,t)-bounded King enters through one of the length-ℓ sides he can only leave through that side again, and such that all squares along the two length-w sides are blocked. Formally, such a fence is a map $F\colon [1\,..\,\ell] \times [1\,..\,w] \to \{0,1\}$, where $F^{-1}(1)$ is the set of blocked squares. The integers ℓ and w are called the* length *and* width *of F, respectively. The* density *of the fence is the ratio*

$$\frac{1}{\ell}\left|\left\{(x,y) \mid 1 \le x \le \ell, 1 \le y \le w, F(x,y) = 1\right\}\right|.$$

The following transformation of an infinite fence into a finite fence is not very difficult.

Lemma 5. *If there exists a periodic infinite (s,t)-fence of density σ then there exist finite (s,t)-fences of the same width w and of density no more than $\sigma + 2w/\ell$ for any length $\ell \ge 1$.*

The $2w/\ell$ term comes from the solid walls to the sides, which are of mere technical relevance. It can always be made arbitrarily small by working with sufficiently long fences, only.

6 A Fractal Fence

Lemma 4 provides us with an infinite fence of density $1 - t/s$, which is strictly smaller than $1/2$ for any α-King with $\alpha < 2$. However, this does not yet suffice to catch any such King, yet. The trick is to assemble many such fences into a huge new fence of slightly smaller density. Iterating this process we will eventually produce fences of arbitrarily small density, which will be very cheap for the Devil to build. The key tool is the following lemma.

Lemma 6. *If there exist finite (s,t)-fences, $s/t \le 2$, of any length above some value ℓ_0, all of the same width w and with density bounded by a common $\sigma < 1/2$, then there also exists a periodic infinite (s,t)-fence with density below $(s/t)\sigma^2$.*

Proof (sketch). The basic idea is to assemble infinitely many identical vertical finite density-σ fences to a wide horizontal fence L. (Requiring only fences longer than a lower bound ℓ_0 is a technical necessity. Because of the solid side walls of size $2w$, very short fences can never have low densities.) As the length ℓ of those finite fences we simply pick any multiple of s larger than ℓ_0 and w, and the gaps between the fences be $m := \lfloor t\ell/s\sigma \rfloor \ge \ell$ squares wide. The width of the big infinite fence L be 7ℓ. The left of Figure 4 shows how the vertical fences are placed in the central ℓ-strip of L. Dashed lines depict the open borders, solid

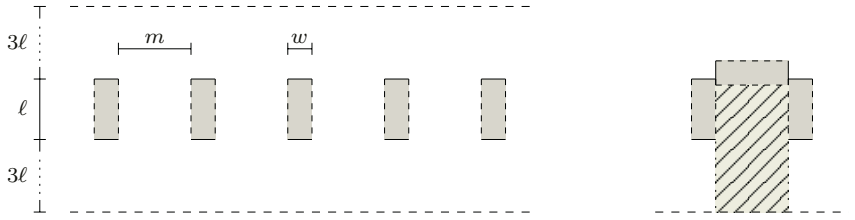


Fig. 4. Assembling many finite vertical fences into one big infinite horizontal fence (left) and blocking a slot (right)

lines the solid side walls. The gray areas are the regions that require permanent Devil play as soon as the King enters if a break though to the other side shall be avoided.

The density of L is easily computed to lie below the required $(s/t)\sigma^2$. Showing that L is indeed an (s,t)-fence is the difficult part. We sketch the key ideas. Assume that the King enters L from the south, so we have to keep him from reaching the upper border. The plan is to build a horizontal fence of length m between the upper ends of two vertical fences whenever the King runs north between them, as indicated in the right of Figure 4. The shaded area there, between two vertical fences and below the (potential) horizontal fence, we call a *slot*. We say that the King is *in standard position* if he is located within a slot whose upper border is already closed or if he sits between two such blocked slots, perhaps within the vertical fence between them.

It is not difficult for the Devil to reach an initial standard position. The first 3ℓ King steps give him enough time to close two or three adjacent slots above the approaching King. The harder part is to keep forcing the King from standard position to standard position as long as he remains in L.

So assume that the King is in standard position. When he enters one of the three surrounding fences, the Devil follows the strategy of that respective fence to make sure that the King does not break through. (Since those fences do not overlap, the Devil is never forced to play in two fences simultaneously.) Hence, the King cannot leave the current slot above line 3ℓ without rebouncing from the fences. If the King leaves the slot that way below, to the left, say, the Devil starts constructing the horizontal fence across the slot to the left. This takes no more than $m\sigma = \lfloor t\ell/s\sigma \rfloor \sigma \le t\ell/s$ Devil moves. During this time the Devil completely ignores the King's play. In particular, he does *not* respond to the possible King's crossing of any fences, thus rendering them ineffective. The clue is that this period of inactive fences is too short for the King to reach any upper fence or the fence to the right of the old slot, and crossing the vertical fence to the left is useless because soon the Devil has the horizontal fence in place there, too. So after the construction, the King will be in standard position again. □

The proof of Theorem 1 is now straight-forward. Pick positive integers s and t with $\alpha \le s/t < 2$, so that the α-King is (s,t)-bounded by Lemma 3. Then Lemma 4 provides us with an infinite periodic (s,t)-fence of density $\sigma < 1/2$ and repeated application of Lemmas 5 and 6 yields fences of smaller and smaller

densities, which converge to zero. In a game against the α-King, the Devil can now arrange four such finite (s,t)-fences of very small density along the four sides of a huge square around the King, who will not be able to reach the boundary of that square before the fences are ready and thus will never be able to leave that prison.

7 Outlook

The immediate open question is, of course, whether the 2-King can be caught—perhaps with the techniques from this paper. This appears probable but observe that we do not have a simple compactness argument by which we could conclude such a statement directly from Theorem 1.

More generally, we hope that α-Kings allow for further small improvements that might bring us gradually closer to new Angel results. On the other hand, it is not unlikely that—in case some Angel is able to escape at all—King speed 2 is already the threshold between winning and losing.

References

1. P. Arnoux. In V. Berthé, S. Ferenczi, C. Mauduit, and A. Siegel, editors, *Substitutions in Dynamics, Arithmetics and Combinatorics*, volume 1794 of *LNM*, chapter 6, pages 143–198. Springer, 2002.
2. Elwyn R. Berlekamp, Hohn H. Conway, and Richard K. Guy. *Winning Ways for your mathematical plays*, volume 2: Games in Particular. Academic Press, 1982.
3. Béla Bollobás and Imre Leader. The Angel and the Devil in three dimensions. Manuscript.
4. John H. Conway. The angel problem. In Richard Nowakowski, editor, *Games of No Chance*, volume 29 of *MSRI Publications*, pages 3–12. 1996.
5. Eric D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In Jiří Sgall, Aleš Pultr, and Petr Kolman, editors, *Proceedings of the 26th Symposium on Mathematical Foundations in Computer Science*, volume 2136 of *LNCS*, pages 18–32, Mariánské Lázně, Czech Republic, 2001.
6. David Gale and Frank M. Steward. Infinite games with perfect information. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games II*, volume 28 of *Annals of Mathematics Studies*, pages 245–266. Princeton University Press, 1953.
7. Martin Gardner. Mathematical games. *Scientific American*, 230(2):106–108, 1974.
8. Thomas Jech. *Set Theory*. Academic Press, 1978.
9. Martin Kutz. *The Angel Problem, Positional Games, and Digraph Roots*. PhD thesis, Freie Universität Berlin, 2004.
http://www.diss.fu-berlin.de/2004/250/ indexe.html.
10. Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

Overlaps Help: Improved Bounds for Group Testing with Interval Queries

Ferdinando Cicalese[1,⋆], Peter Damaschke[2,⋆⋆], Libertad Tansini[2], and Sören Werth[3]

[1] AG Genominformatik, Technische Fakultät, Universität Bielefeld, Germany
nando@cebitec.uni-bielefeld.de
[2] School of Computer Science and Engineering, Chalmers University, 41296 Göteborg, Sweden
{ptr,libertad}@cs.chalmers.se
[3] Mathematisches Seminar Bereich 2, University Kiel, 24118 Kiel, Germany
swe@numerik.uni-kiel.de

Abstract. Given a finite ordered set of items and an unknown distinguished subset P of up to p positive elements, identify the items in P by asking the least number of queries of the type "does the subset Q intersect P?", where Q is a subset of consecutive elements of $\{1, 2, \ldots, n\}$. This problem arises e.g. in computational biology, in a particular method for determining splice sites. We consider time-efficient algorithms where queries are arranged in a fixed number s of stages: in each stage, queries are performed in parallel. In a recent paper we devised query-optimal strategies in the special cases $p = 1$ or $s = 2$, subject to lower-order terms. Exploiting new ideas we are now able to provide a much neater argument that allows doubling the general lower bound for any $p \geq 2$ and $s \geq 3$. Moreover, we provide new strategies that match this new bound up to the constant of the main term. The new query scheme shows an effective use of overlapping queries within a stage. Remarkably, this contrasts with the known results for $s \leq 2$ where optimal strategies were implemented by disjoint queries.

1 Introduction

Group testing is a basic paradigm in the theory of combinatorial search: The *positive* members of a set of objects O are to be determined by asking as few queries as possible of the form "does the subset $Q \subseteq O$ contain at least one positive object?". The main idea is to test for positivity of entire groups as opposed to individually searching for each positive - a main advantage being that a negative answer to a query gives information that all items belonging to it are negative, i.e., non-positive, and then they can be ignored in the following.

⋆ Supported by the Sofja Kovalevskaja Award of the Alexander von Humboldt Foundation

⋆⋆ Supported by the Swedish Research Council (Vetenskapsrådet), project "Algorithms for searching and inference in genetics", grant no. 621-2002-4574

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 935–944, 2005.

Group testing methodologies have proved to be useful in a variety of situations such as quality control, multiple access communication, computational molecular biology, data compression, and Data Streams algorithms among the others (see [1, 3, 4, 6, 9, 10]).

Problem Statement. In this paper we consider the variant of group testing arising when the search space is a linearly ordered set and the queries are constrained to be intervals of this set. This is *Interval Group Testing*. More precisely, an instance of the problem is given by two non negative integers p and n and a subset $P \subseteq O = \{1, 2, \ldots, n\}$, such that $|P| \leq p$. The set O is the search space and P is the set of *positive* objects that have to be identified. Queries (tests) are constrained to be intervals $\{i, i+1, \ldots, j-1, j\}$, for some $i, j \in \{1, 2, \ldots, n\}$. The target is to identify P by using the minimum possible number of queries. We assume that tests are arranged in *stages*: in each stage a certain number of tests is performed non-adaptively, while tests of a given stage can be determined depending on the outcomes of the tests in the previous stages. For each value of the parameters n, p, s, we want to determine $\mathcal{N}(n, p, s)$, the worst-case minimum number of tests that are necessary (and sufficient) to successfully identify all positives in a search space of cardinality n, under the hypothesis that the number of positives is at most p and s-stage algorithms are used.

Our Results. In this paper we shall concentrate on s-stage interval group testing procedures - in particular for all $s \geq 3$. We determine the asymptotic for $N(n, p, s)$ up to the exact evaluation of the constant of the main term. In particular, we prove a new lower bound on $\mathcal{N}(n, p, s)$ that doubles the constant of the main term with respect to the previously published results. Moreover, we show that such bound can be matched up to that constant. A remarkably new feature of our strategies is the use of overlapping queries within a stage in an effective manner. This partially explains the difficulties encountered in the attempt to generalize to the case $s \geq 3$ the techniques employed for the case $s \leq 2$ where optimal strategies are implementable with disjoint queries.

Motivations and Related Research. Interval Group Testing naturally arises in the problem of determining exon-intron boundaries within a gene [7, 8]. In a very simplified model, a gene is modeled as a collection of disjoint substrings within a long string representing the DNA molecule. These substrings are called *exons*, and the substrings separating them are called *introns*. Only the concatenation of exons codes for a protein whilst the biological role of introns is rather unclear. Each boundary point linking an exon and an intron is called a *splice site*. The determination of splice sites is often a critical point in searching for mutations associated with a gene responsible for a disease, because only mutations in exons are relevant.

A typical approach to the splice sites identification problem consists in comparing the original *genomic* DNA to the cDNA obtained in laboratory. cDNA is a "purified" version of the genomic DNA, in which all introns have been removed and the exons spliced together. In [8] a new experimental protocol is proposed that searches for the exons boundaries in the cDNA using group testing. In fact,

using standard experimental procedures (polymerase chain reaction, PCR) one can select two positions in the cDNA string and determine whether they are at the same distance as they were in the original genomic DNA string. If these distances do not coincide then at least one intron (and hence a splice site) must be present in the genomic DNA between the two selected positions. The formulation of splice sites identification as a group testing problem with interval queries is explicitly stated in [5, 7, 8], where an additional constraint on the queries sizes takes into account technological limitations of the PCR procedures.

The work [8] and the book [7] report about the experimental evaluation, on real data, of the algorithm ExonPCR, that finds exon-intron boundaries within a gene. The authors of [8] give also a simple asymptotic analysis of their $\Theta(\log n)$-stage algorithm. In [2] the first rigorous algorithmic study of Interval Group Testing was presented and for the case $s \leq 2$ a precise evaluation of $\mathcal{N}(n, p, s)$ was given.

2 Preliminaries: Basic Results

In [2] optimal 1-stage and 2-stage interval group testing algorithms were fully characterized. Moreover, for the case $p = 1$, s-stage optimal algorithms were also presented. For later reference in the following three facts we summarize the main results of [2] on multistage interval group testing.

Proposition 1 (1-stage). *[2] For all non-negative integers n, p, it holds that*

$$\mathcal{N}(n, p, 1) = \begin{cases} \lceil (n+1)/2 \rceil & if \quad p = 1, \\ n & otherwise. \end{cases}$$

Fact 1 says that in 1-stage (non-adaptive) interval group testing the size of the optimal strategy is linear in the size of the search space. This negative result gives more motivation to the study of multistage algorithms.

Definition 1. *For any integer $s \geq 2$, an s-stage interval group testing algorithm $\mathcal{A}$ consists of s successive 1-stage algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_s$, confining the positives in a collection of smaller and smaller subintervals. The last stage returns subintervals of size one since it determines the exact positions of the positives.*

Let $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_s)$ be an s-stage interval group testing algorithm. For $j = 1, 2, \ldots, s$, we let $S(\mathcal{A}_j)$ be the collection of disjoint maximal intervals, henceforth called segments*, that each contain at least one positive element as a result of the tests performed in the first j stages. We let $S(\mathcal{A}_0) = \{\{1, ..., n\}\}$.*

For s-stage interval group testing algorithm *for at most one positive* we have the following.

Proposition 2 (s-stage for 1 positive). *[2] Fix an integer $s \geq 1$. For all integers $n > 2^s - 1$, we have $\mathcal{N}(n, 1, s) = \frac{s}{2} n^{1/s} + O(s)$.*

As opposed to the case of one positive, for multistage interval group testing with $p \geq 2$ no exact results are known. A natural strategy consisting in asking disjoint queries in each stage turns out to be optimal for the particular case $s = 2$. The resulting characterization is captured by the following.

Proposition 3 (2-stage for more positives). *[2] For $p = o(\sqrt{n})$, the 2-stage interval group testing problem for at most p positives needs $\mathcal{N}(n, p, 2) = 2\sqrt{p-1}\sqrt{n} + O(p)$ queries, and they are also sufficient.*

Unfortunately, the attempt to extend to the case $s \geq 3$ the techniques employed in the proof of proposition 3 hasn't been successful.

In [2] the following general result was given and closing the gap between the lower and upper bounds was left as a major open problem.

Proposition 4 (s-stage for $p \geq 2$ positives). *[2] For all non-negative integers n, s, p such that $n \geq 2^s(p-1)$, we have*

$$\frac{ps}{4}\sqrt[s]{n/p} \leq \mathcal{N}(n,p,s) \leq s(p-1)^{(s-1)/s}(n-(n \bmod (p-1)))^{1/s}+s+p+(n \bmod (p-1))-2.$$

The upper bound is by a slight modification of the natural strategy consisting in asking, in each stage j, in each segment in $S(\mathcal{A}_{j-1})$, $\sqrt[s]{n/p}$ disjoint queries evenly distributed. In the last stage the algorithm simply searches exhaustively the segments returned by the previous phase.

3 Lower Bound for Multistage Interval Group Testing

For ease of presentation, in the following two sections, we use simplifications. For example, we shall neglect rounding. It will always be clear that these inaccuracies do not affect the asymptotic results.

We identify every element with a unit interval, so that query intervals have endpoints with integer coordinates.

A left (right) endpoint of a query interval is also called a left (right) *bracket*. Let F be a family of q intervals queried in a single stage of an algorithm. We will consider sets of identical endpoints of intervals from F as sequences of brackets with infinitesimal distances, i.e. without elements in between. The ordering is arbitrary but fixed. Therefore, we will assume that the q intervals in F have $2q$ different endpoints (brackets).

A *piece* is an interval flanked by two endpoints of some intervals from F, without other such brackets in between. Hence, a family F of q interval queries always partition the search space into exactly $2q + 1$ pieces, either of positive integer length or of infinitesimal length.

Our lower bound proof is based on an adversary strategy. Given an arbitrary s-stage algorithm, our adversary behaves as follows. First of all, the search space is split in p intervals of length n/p called *segments*. In every stage, one positive element is placed in a longest piece in every segment. The adversary answers consistently. To simplify the argument, the adversary also reveals the pieces that

contain positives. (Giving additional information can only improve the searcher's situation.) In the next stage, the new segments are these pieces.

Note that the adversary's answering strategy is well defined, since the exact locations of positive elements within the pieces need not be fixed. We will derive a lower bound on the total number of queries needed to reduce all p segments to single elements, in this constrained (for the adversary) setting.

Theorem 1. *Fix non negative integers* $n, p \geq 2, s \geq 3$. *Then,* $\mathcal{N}(n,p,s) \geq \frac{1}{2}ps(\sqrt[s]{n/p} - 1)$ *queries.*

Proof. Recall that our adversary places exactly on positive element in a longest piece of the jth segment, and this segment becomes the jth segment in the next stage, for $j = 1, \ldots, p$. The searcher *must* reduce the size of the jth segment from n/p to 1 in s stages.

Consider any fixed j. We analyze the necessary number b of brackets used in the jth segment during all stages. First of all, we can assume that the brackets in every stage split the current segment in pieces of equal size, since this minimizes the size of the longest piece.

Let b_i be the number of brackets used in stage i in the considered segment, hence $\sum_{i=1}^{s} b_i = b$. By the previous observation, the segment length is reduced by factor $\prod_{i=1}^{s}(b_i + 1)$ after s stages. For any fixed "budget" b of brackets, this product is maximized if all b_i are equal, due to an elementary fact. On the other hand, this product has to be at least n/p. Together this implies $b \geq s(\sqrt[s]{n/p} - 1)$. Since the argument applies to every j, the total number of brackets is p times as large. Finally note that every query interval has two brackets. □

A more elaborated analysis might be able to remove the negative lower-order term.

4 A New Strategy for Multistage Interval Group Testing

In this section we present a strategy for s-stage interval group testing that matches the lower bound presented in the previous section, as s grows.

A remarkable feature of this new strategy is the use of overlapping queries in an effective manner. From the analysis we present, it seems that overlapping queries are necessary for the optimality, as s grows. This is quite surprising since it apparently contrasts with the optimal results for the case $s = 2$ where optimal strategies are implementable by disjoint queries. Still the fact that, asymptotically, optimality requires overlapping explains the difficulties encountered in to attempt to extend straightforwardly the results of [2] to the case $s \geq 3$.

The intuition behind our strategy can be outlined as follows. There is an obvious strategy with disjoint and equally long query intervals in every stage, that needs $ps\sqrt[s]{n/p}$ queries. Now we extend the query intervals to both sides, so that they overlap. This makes the pieces between these overlaps shorter, and our hope is that the positives are in these non-overlapping pieces. The difficult case is when two intersecting query intervals say *yes*. Then, there could

be positives in the intersection, but also in both "wings", i.e., in the neighbored pieces. Altogether we have to explore an interval that is longer than in the obvious strategy. However, if both wings actually contain positives, we will detect at least one new positive. The crucial idea is now to start the search for more positives on the wings with fewer (but longer) intervals, compared to the obvious strategy. This saves queries, but incurs extra costs to every new positive (because we have to find it within a longer start interval). However, we have these extra costs only once for every positive, and then we continue normally. If we choose the lengths accordingly, we still get an improvement upon the obvious strategy. (One may imagine that the extra costs are evenly distributed to the s stages.)

After this sketch of the idea, we formally describe the two different query schemes our strategy relies on.

Definition 2. *Given an interval $[a, b]$ of length ℓ and numbers u and α, an α-query scheme of unit u on $[a, b]$ is a set of $\frac{\ell}{u(1+\alpha)}$ overlapping queries*[1] *covering $[a, b]$ defined as follows. The first*[2] *query is of length $1 + \alpha$. The next queries, until the last but first, are of length $(1 + 2\alpha)u$ and they overlap the previous and the next by αu. The last query is either of length $(1 + \alpha)u$ or of length αu and it overlaps the previous query by αu.*

Definition 3. *Given an interval $[a, b]$ of length ℓ and numbers u and λ, a λ-query scheme of unit u on $[a, b]$ is a set of $\frac{\ell}{\lambda u}$ disjoint queries. More precisely, for $j = 1, \ldots, \frac{\ell}{\lambda u}$,* Query j *covers the interval $[a + (\lambda u)(j - 1), (a - 1) + (\lambda u)j]$ (its length is λu).*

Figure 4.1 contains an example of both query schemes.

For each $i = 1, \ldots, s$ we define the unit length of our query schemes for stage i as the value $u_i := \frac{n}{p\left(\sqrt[s]{n/p}\right)^i}$. In each stage our strategy makes use only of questions organized in α-query schemes and λ-query schemes of unit u_i.

Answers assign weights to queries and pieces according to the following scheme: A query in an α-scheme gets weight 2 if it answers *yes* and the intersecting queries answer *no*. Otherwise it gets weight 1. A query in a λ-scheme has weight 2.

A piece π has weight $w(\pi)$ equal to the sum of the queries it intersects.

Note that the only possible weights for a piece in $S(\mathcal{A}_i)$ will be 1 and 2. As an example, consider the α-querying schemes of stage 1 in Figure 4.1. Suppose only the queries answer *yes*. Thus, only the first three pieces on the left and one on the very right are selected for the next stage. In particular, the first and the third piece from the left (both of length u_1) get weight 1. The second piece from the left (of length αu_1) gets weight 2, as well as the last piece on the right.

Our algorithm is presented in Figure 4.2. We show it parameterized on the choice of the values α and λ.

[1] Here we are neglecting rounding again. But this might imply at most an additive factor of ps to the upper bound we shall prove.

[2] We assume the queries numbered from left to right.

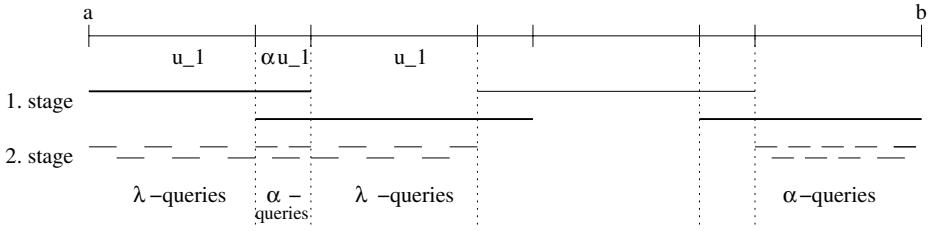


Fig. 4.1. Stage 2 - α and λ-schemes

$P_0 = \{[a,b]\}$, $w([a,b]) = 2$
Algorithm $\mathcal{A}_i$, $(i = 1, 2, \ldots, s-1)$
If $|S(\mathcal{A}_{i-1})| = p$
  For each segment $\sigma \in S(\mathcal{A}_{i-1})$ in parallel
    Run an optimal $s - i + 1$-stage algorithm for one positive on σ.
Otherwise
  Let P_i be the collection of pieces in $S(\mathcal{A}_{i-1})$
  for each piece $\pi \in P_i$
    if $w(\pi) = 1$, ask queries in π according to λ-query scheme of unit $\left(\frac{n}{p}\right)^{1-\frac{i}{s}}$
      Else ask queries in π according to α-query scheme of unit $\left(\frac{n}{p}\right)^{1-\frac{i}{s}}$
Algorithm $\mathcal{A}_s$
If $|S(\mathcal{A}_{i-1})| = p$
  For each segment $\sigma \in S(\mathcal{A}_{i-1})$ in parallel
    Run an optimal 1-stage algorithm for one positive on σ.
Otherwise
  For each object x in the segments of $S(\mathcal{A}_{i-1})$
    Ask the query $\{x\}$
  Return the objects that answered *yes*

Fig. 4.2. The algorithm $\mathcal{A}$

We have the following result.

Theorem 2. *Fix non-negative integers $n, s \geq 3, p$. Then, there exists α and λ such that the s-stage interval group testing algorithm $\mathcal{A} = \mathcal{A}_1, \ldots, \mathcal{A}_s$ finds a set of $\leq p$ positives in a set of cardinality n using at most*

$$ps\left(\frac{1}{2} + \sqrt{\frac{p-1}{sp}}\right)\sqrt[s]{\frac{n}{p}} + 2(p-1)\sqrt{\frac{sp}{p-1}}\sqrt[s]{\frac{n}{p}} + \left(2 + \frac{\sqrt{sp} - 2\sqrt{p-1}}{\sqrt{sp} + 2\sqrt{p-1}}\right)\sqrt[s]{\frac{n}{p}}$$

queries.

Proof. Let us first assume that in each stage $j = 1, 2, \ldots, s-1$, we have $|S(\mathcal{A}_j| \leq p-1$. Therefore the algorithm will never make use of the optimal procedure for interval group testing with one positive. We shall later show that, conversely, when this happens the algorithm can only save questions, so preserving our bound.

Recall that $S(\mathcal{A}_0) = \{\{1, 2, \ldots, n\}\}$. For $j = 1, 2, \ldots, s$, each segment in $S(\mathcal{A}_j)$ is of one of the following types:

1. Segments consisting of a single piece of length u_j. These are the ones produced by a query in an α-scheme that answers *yes* whilst all the queries it intersects answer *no*.
2. Segments consisting of a sequence of pieces of alternating lengths u_j and αu_j, starting and ending with pieces of length u_j. These are the segments produced when 2 or more contiguous queries in an α-scheme answer *yes*.
3. Segments consisting of a single piece of length λu_j. This type of segments is produced when a query in a λ-scheme answers *yes*. We denote the number of the segments of this type in $S(\mathcal{A}_j)$ by l_j.

We assume that the only segment in $S(\mathcal{A}_0)$ is of type 1.

Let q be the number of queries used in a run of the algorithm. Let $\mathcal{Q}$ be the set of such queries. Assume that the positives are labeled from 1 to p. For each $j = 1, 2, \ldots, s$ and for each $k = 1, 2, \ldots, p$, let $\mathcal{Q}_{jk}$ be the set of questions among the above q that in stage j were used[3] to acquire information about the position of the k-th positive. Let $q_{jk} = |\mathcal{Q}_{jk}|$.

Since, in general a question can give information about more than one positive, we immediately have $q = |\mathcal{Q}| = |\bigcup_{k=1}^{p} \bigcup_{j=1}^{s} \mathcal{Q}_{jk}| \leq \sum_{k=1}^{p} \sum_{j=1}^{s} q_{jk}$.

We have the following claims:

Claim 1. For each $j = 0, 2, \ldots, s-1$ $k = 1, 2, \ldots, p$ and $i = 1, 2, 3$, let $\delta_{jk}^{[i]}$ be 1 if in $S(\mathcal{A}_j)$ the kth positive is in a piece of a segment of type i. Otherwise, let $\delta_{jk}^{[i]} = 0$. Then, for $j = 1, 2, \ldots, s-1$, and $k = 1, 2, \ldots, p$,

$$q_{jk} \leq \left(\delta_{j-1k}^{[1]} \frac{1}{1+\alpha} + \delta_{j-1k}^{[2]} \left(\frac{\alpha}{1+\alpha} + \frac{2}{\lambda} \right) + \delta_{j-1k}^{[3]} \frac{\lambda}{1+\alpha} \right) \sqrt[s]{\frac{n}{p}}$$

and $\quad q_{sk} \leq \left(\delta_{s-1k}^{[1]} + \delta_{s-1k}^{[2]} (\alpha + 2) + \delta_{s-1k}^{[3]} \lambda \right) \sqrt[s]{\frac{n}{p}}$.

This follows from the rules employed by the algorithm. We shall now clarify the computation of the coefficients of the $\delta_{jk}^{[i]}$'s. The coefficient of $\delta_{jk}^{[1]}$ is the number of questions performed to apply an α-query scheme to pieces of segments of type 1. In $S(\mathcal{A}_{j-1})$ segments of type 1 consist of one piece of size u_{j-1}. The algorithm applies in such a segment an α-query scheme of unit u_j. Thus, an easy computation shows that the algorithm performs in this case $\frac{1}{1+\alpha} \sqrt[s]{n/p}$ queries. The coefficient of the $\delta_{jk}^{[3]}$ can be similarly obtained. For computing the coefficient of $\delta_{jk}^{[2]}$, we observe that a positive in a segment of type 2 can (in the worst case) produce queries in one of the pieces of length αu_{j-1} and in the two surrounding pieces of length u_{j-1} [4]. This is the case when the positive is in the intersection of two queries of an α-scheme.

[3] Of course this is an *a posteriori* assignment of questions to positives that we do for the analysis.

[4] Note also that this association of pieces to positives is enough to cover all pieces, of a segment of type 2, where new queries will be asked.

Finally, to obtain the bound on q_{sk}, it is sufficient to consider that in the last stage the algorithm searches exhaustively the segment in $S(\mathcal{A}_{s-1})$ that tested positively because of the k-th positive. The coefficients of the δ's are just the sizes of the three types of pieces possibly returned by the $s-1$-th stage.

Claim 2. We have that $\sum_{j=1}^{s-1} l_j \leq 2(p-1)$.
Recall that l_j counts the number of segments of type 3. These are generated by *yes* answers to queries in a λ-scheme. A λ-scheme always follows an α-scheme where two intersecting queries have answered *yes*. A moment's reflection shows that a *yes* answer to a query in a λ-scheme indicates that the *yes*-queries in the preceding α-scheme had split the set of positives into at least two parts. Note that for a set of p positives there can be at most $p-1$ steps in which the set is divided in an α-scheme and the two parts of these partitions are identified with the next λ-schemes. Hence, we have the desired bound.

Claim 3. For each $i = 1, 2, 3$, let $\mathcal{Q}_{jk}^{[i]}$ be the subset of $\mathcal{Q}_{jk}$ whose query are asked in a piece of a segment of type i. We have that $|\bigcup_{k=1}^{p}\bigcup_{j=1}^{s}\mathcal{Q}_{jk}^{[3]}| = \sum_{j=1}^{s-1} l_j$.

Putting together the above claims we have:

$$
\begin{aligned}
q = |\mathcal{Q}| &= |\bigcup_{k=1}^{p}\bigcup_{j=1}^{s}\mathcal{Q}_{jk}| = \left|\bigcup_{k=1}^{p}\bigcup_{j=1}^{s}\bigcup_{i=1}^{3}\mathcal{Q}_{jk}^{[i]}\right| \\
&\leq \left(\sum_{k=1}^{p}\sum_{j=0}^{s-2}\left(\delta_{jk}^{[1]}\frac{1}{1+\alpha}+\delta_{jk}^{[2]}\left(\frac{\alpha}{1+\alpha}+\frac{2}{\lambda}\right)\right)+\left(\delta_{s-1k}^{[1]}+\delta_{s-1k}^{[2]}(\alpha+2)\right)+\sum_{j=1}^{s-1} l_j\lambda\right)\sqrt[s]{\frac{n}{p}} \\
&\leq \left(\sum_{k=1}^{p}\sum_{j=0}^{s-2}\left(\delta_{jk}^{[1]}\frac{1}{1+\alpha}+\delta_{jk}^{[2]}\left(\frac{\alpha}{1+\alpha}+\frac{2}{\lambda}\right)\right)+\left(\delta_{s-1k}^{[1]}+\delta_{s-1k}^{[2]}(\alpha+2)\right)+2(p-1)\lambda\right)\sqrt[s]{\frac{n}{p}} \\
&\leq \left(\sum_{k=1}^{p}\sum_{j=0}^{s-2}\left(\delta_{jk}^{[1]}\frac{1}{1+\alpha}+\delta_{jk}^{[2]}\left(\frac{\alpha}{1+\alpha}+\frac{2}{\lambda}\right)+\frac{2(p-1)}{p}\frac{\lambda}{s}\right)+\left(\delta_{s-1k}^{[1]}+\delta_{s-1k}^{[2]}(\alpha+2)\right)\right)\sqrt[s]{\frac{n}{p}}.
\end{aligned}
$$

The values of the parameters α and λ that minimize the last expression for all possible choices of the variables $\delta_{jk}^{[i]}$, are given by

$$\lambda = \sqrt{\frac{sp}{(p-1)}} \quad \textit{and} \quad \alpha = \frac{\lambda-2}{\lambda+2}.$$

which gives the desired result for $s \geq 4$.

Notice that, in fact, for $s = 3$, $p \geq 4$ the above value for λ would imply that $\alpha \leq 0$. As a matter of fact, when $s = 3$, we have a stronger version of Claim 2 which leads to a bound even slightly better. Due to the space constraints we shall omit the proof of this case in this extended abstract.

References

1. G. Cormode, S. Muthukrishnan, *What's hot and what's not: Tracking most frequent items dynamically*, in: ACM Principles of Database Systems, 2003
2. F. Cicalese, P. Damaschke, U. Vaccaro, *Optimal group testing strategies with interval queries and their application to splice site detection*, to appear in Proc. of the 2005 Int. Workshop on Bioinformatics Research and Applic. (IWBRA), 2005. Journal version to appear in Int. Journal of Bioinformatics Research and Applications.
3. D.Z. Du and F.K. Hwang, *Combinatorial Group Testing and its Applications*, World Scientific, Singapore, 2000.
4. M. Farach, S. Kannan, E.H. Knill, S. Muthukrishnan, *Group testing with sequences in experimental molecular biology*, in: Proc. of Compression and Complexity of Sequences 1997, B. Carpentieri, A. De Santis, U. Vaccaro, J. Storer (Eds.), IEEE CS Press, pp. 357-367, 1997.
5. R. Karp, *ISIT'98 Plenary Lecture Report: Variations on the theme of 'Twenty Questions'*, IEEE Information Theory Society Newsletter, vol. **49**, No.1, March 1999.
6. E. H. Hong and R.E. Ladner, *Group testing for image compression*, IEEE Transactions on Image Processing, **11(8)**, pp. 901-911, 2002.
7. P.A. Pevzner, *Computational Molecular Biology, An Algorithmic Approach*, MIT Press, 2000.
8. G. Xu, S.H. Sze, C.P. Liu, P.A. Pevzner, N. Arnheim, *Gene hunting without sequencing genomic clones: Finding exon boundaries in cDNAs*, Genomics, **47**, pp. 171-179, 1998.
9. Hung Q. Ngo and Ding-Zhu Du, *A survey on combinatorial group testing algorithms with applications to DNA library screening*, in: *Discrete Mathematical Problems with Medical Applications*, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., **55**, Amer. Math. Soc., pp. 171-182, 2000.
10. J. Wolf, *Born again group testing: Multiaccess communications*, IEEE Trans. Information Theory, **IT-31**, pp. 185-191, 1985.

New Efficient Simple Authenticated Key Agreement Protocol

Eun-Jun Yoon and Kee-Young Yoo*

Department of Computer Engineering, Kyungpook National University
Daegu 702-701, Republic of Korea
ejyoon@infosec.knu.ac.kr, yook@knu.ac.kr

Abstract. Recently, Kim et al. proposed an improvement to the Simple Authenticated Key Agreement (SAKA) protocol that has the same stability as the existing methods, and that has a much efficient processing performance. However, this improved scheme is still susceptible to off-line password guessing attacks and the integrity violence of a session key from illegal modification. The current paper demonstrates the vulnerability of Kim et al.'s scheme to off-line password guessing attacks and the integrity violence of the session key from illegal modification, and then presents an improved protocol based on the elliptic curve discrete logarithm problem (ECDLP) to resolve such problems. As a result, the proposed protocol resists off-line password guessing attacks and modification attack, while also providing more security and efficiency.

Keyword: Authenticated key agreement, Password guessing attack, Modification attack, Elliptic curve discrete logarithm problem

1 Introduction

The Dffie-Hellman key agreement scheme [1] was proposed as a solution to produce a common session key between two parties, and it is seen as an epochal breakthrough that can produce a common session key without any prior common information. However, this method has a weakness of possible man-in-the middle attacks [2].

There have been several methods to solve this problem, such as key exchange protocol using certificates [3] and the authenticated key exchange protocol in which two parties share a secret password (pre-shared password) beforehand [4, 5]. The former has a weakness in that it needs a trusted third certification party. In such a system, if the number of users is increased, larger storage for saving the user's certification and higher network bandwidth due to the verification of digital signature are needed. Therefore, it is difficult to extend the system including the key exchange protocol. In addition, there is a weak point in that this system undergoes negative influences if any point of certification processes, and two communication parties share a secret password (pre-shared

* Corresponding author: Tel.: +82-53-950-5553; Fax: +82-53-957-4846

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 945–954, 2005.

password) before initiation. Thus, the safety of the system depends on each user, not on a third certification party. Recently, Seo and Sweeney proposed a new key agreement protocol based on the Diffie-Hellman protocol called the simple authenticated key agreement algorithm (SAKA). In the SAKA protocol, two parties have a pre-shared password for data communication, produce a session key by exchanging messages, and confirm each other. Because of the advantages that can simplify key agreement, SAKA-like protocols are widely used in research on key agreement, and therefore there are numerous articles [5–14] to continuously enhance SAKA-like protocols.

In 2004, Kim et al. [14] also proposed an improvement to the SAKA protocol that has the same stability as the existing methods, but that possesses a much more efficient processing performance. However, this improved scheme is still susceptible to the off-line password guessing attacks and the integrity violence of the session key from illegal modification. Accordingly, the current paper demonstrates the vulnerability of Kim et al.'s scheme to off-line password guessing attacks and the integrity violence of the session key from illegal modification, and then presents a more efficient and simple key agreement protocol (ESAKA) based on the elliptic curve discrete logarithm problem (ECDLP) to resolve such problems. The Elliptic Curve Cryptosystem (ECC) presents an attractive alternative cryptosystem because its security is based on the elliptic curve discrete logarithm problem (ECDLP). ECC operates over a group of points on an elliptic curve and offers a level of security comparable to classical cryptosystems that use much larger key sizes. As a result, the proposed ESAKA protocol resists off-line password guessing attacks and the integrity violence of the session key from illegal modification, while also providing more security and efficiency.

The remainder of this paper is organized as follows: Section 2 briefly reviews their protocol and demonstrates an off-line password guessing attacks and modification attack on Kim et al.'s protocol. The proposed ESAKA protocol is presented in Section 3, while Section 4 discusses the security and efficiency of the proposed protocol. Final conclusions are given in Section 5.

2 Cryptanalysis of Kim et al.'s Protocol

This section briefly reviews Kim et al.'s SAKA protocol [14] and then shows the security flaws of their protocol. Notations used in Kim et al.'s protocol and proposed protocol are defined as follows:

- A, B: two communicating parties;
- C: an attacker;
- id_A, id_B: the identities of A and B;
- n: a large prime number;
- g: a generator $\in Z_n^*$ with the order $n-1$;
- P: the common password shared between A and B;
- Q: an integer computed from P;
- Q^{-1}: the inverse of $Q \pmod{n-1}$;

- $E(GF_q)$: an additive group of points on an elliptic curve E over a finite field $GF(q)$;
- G: the generating element(point) of $E(GF_q)$ under consideration, $GF(q)$;
- P^*: an elliptic curve point $\in E(GF_q)$ computed from P;
- a: a secret random integer chosen by A;
- b: a secret random integer chosen by B;
- $H(\cdot)$: a secure one-way hash function, where H: $\{0,1\}^* \rightarrow \{0,1\}^k$, e.g. $H(x)$ is a secure hash function at the x-coordinate of point $X \in E(GF_q)$.

2.1 Review of Kim et al.'s Protocol

Figure 1 illustrates Kim et al.'s SAKA-like protocol. Their protocol proceeds with the following 4 steps:

1. A chooses a random number a, computes $X_1 = g^{aQ} \bmod n$, and sends X_1 to B.
2. B chooses a random number b, computes Y_1 and X, where $Y_1 = g^{bQ} \bmod n$ and $X = X_1^{Q^{-1}} = g^a \bmod n$. Then, sends back X and Y_1 to A.
3. A verifies if X is equal to $g^a \bmod n$. If they match each other, A authenticates B. Then, A computes $Y = Y_1^{Q^{-1}} = g^b \bmod n$ and the session key $K_1 = Y^a = g^{ab} \bmod n$. Then, A sends Y to B.
4. B verifies if Y is equal to $g^b \bmod n$. If they match each other, B authenticates A. Then, B computes the session key $K_2 = X^b = g^{ab} \bmod n$.

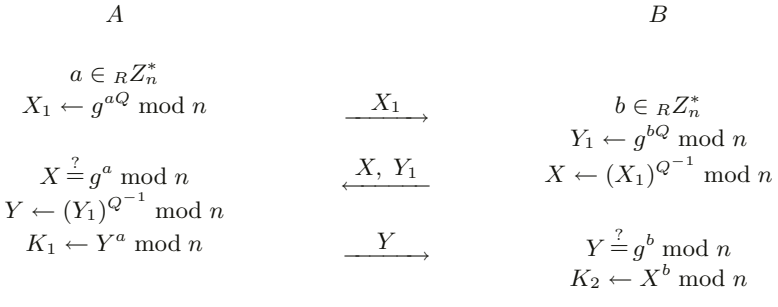


Fig. 1. Kim et al.'s SAKA protocol

2.2 Off-Line Password Guessing Attack

Unfortunately, Kim et al.'s protocol is vulnerable to off-line password guessing attack. An attacker C who can capture messages exchanged over network can easily obtain a legitimate communication parties' password P. The attack proceeds as follows:

Step 1*. An attacker C records a pair of information $\{X_1, X\}$ exchanged in a valid key agreement session. It is easy to obtain the information since it is readily available over the open network.
Step 2*. In order to obtain the password P shared two legitimate parties, the attacker C makes a guess at the secret password P^* and derives corresponding Q^* and $Q^{*-1} \bmod n-1$.
Step 3*. Computes X^{Q^*} and checks if $X_1 = X^{Q^*} \bmod n$, where X_1 and X are the information that he or she captured.
Step 4*. If it is not correct, the attacker C repeatedly performs it until $X_1 = X^{Q^*} \bmod n$.

Furthermore, if an attacker C records an information pair $\{Y_1, Y\}$ exchanged in a valid key agreement session, then the attacker can also obtain the password P shared by two legitimate parties as described above. In fact, attacker C only can get Q instead of P from the above attack. Thus, the damage of the attack is limited to one session. However, suppose $f(\cdot)$ is a function to get an integer Q computed from P; that is, $Q = f(P)$. To get password P, the attacker C must break function $f(\cdot)$ since the function $f(\cdot)$ is public, meaning that its description is known, and that anyone can compute it.

Unlike typical private keys, the password has limited entropy, constrained by the memory of the user. Roughly speaking, the entropy of human memorable passwords is about 2 bits per character. Therefore, an attacker's goal of obtaining a legitimate communication parties' password can be achieved within a reasonable time. Thus, the off-line password guessing attack in Kim et al.'s protocol should be considered as the realistic one.

2.3 Integrity Violence of the Session Key from Illegal Modification

Kim et al.'s protocol is also vulnerable to an integrity violence of the session key from illegal modification. Suppose that attacker C interposes the communication between A and B. Then, attacker C can perform the illegal modification attack as follows:

Step 1*. Upon intercepting message $X_1 = g^{aQ} \bmod n$ sent by A, the attacker can replace it with $(X_1)^t = g^{aQt} \bmod n$, where t is a randomly chosen integer.
Step 2*. Similarly, upon intercepting message $\{X, Y_1\}$ sent back by B, the attacker can replace X with $(X)^{t^{-1}} \bmod n = g^a \bmod n$ and Y_1 with $(Y_1)^t = g^{bQt} \bmod n$, respectively.

After all, A and B can compute the same wrong session key $K_1' = K_2' = g^{abt} \bmod n$, respectively. However, A and B cannot detect the generation of this wrong session key because they have the same session key. From now, A and B shall use the wrong session key in encrypting/decrypting their messages. Through this illegal modification attack, attacker C can neither obtain K_1' nor K_2' but can make two parties believe and use an unintended session key. In fact,

an illegal modification attack is not a serious attack, since it cannot prevent the two communication parties from reaching a common secret key, even though this key is not the correct one. Most important, the attacker cannot access the agreed common key from this illegal modification attack. However, since the Diffie-Hellman session key g^{ab} is invalid, it cannot guarantee the integrity of the session key.

3 ESAKA Protocol

This section proposes a much simpler protocol based on ECDLP that, unlike Kim et al.'s protocol, can withstand the off-line password guessing attack and the integrity violence of the session key from illegal modification. The proposed protocol can gain benefits from the key block size, speed, and security. The weakness of Kim et al.'s protocol is due to the two values X_1 and X in their key establishment and key validation phase, respectively. Since the values are publicly visible, an attacker capturing them can easily guess legitimate communication parties' passwords by judging the correctness of the guess. Thus, the most important requirement to prevent a guessing attack is to eliminate the information that can be used to verify the correctness of the guess. The main idea of our scheme is to isolate such information by using an asymmetric structure in the messages exchanged. Figure 2 illustrates the proposed ESAKA protocol based on ECDLP. A and B choose an elliptic curve E over a finite field $GF(q)$. A and B only need to store a mutual password. The proposed ESAKA protocol proceeds with the following 4 steps:

1. A chooses a secret random integer a, computes $X_1 = a \cdot G + P^*$, and sends X_1 to B.
2. B chooses a secret random integer b, and computes Y_1, X and SK_B as follows:

$$\begin{aligned} Y_1 &= b \cdot G, \\ X &= X_1 - P^* = a \cdot G, \\ SK_B &= b \cdot X = a \cdot b \cdot G. \end{aligned} \tag{1}$$

   Then, B sends back Y_1 and $H(X, SK_B)$ to A.
3. A computes SK_A as follows:

$$SK_A = a \cdot Y_1 = a \cdot b \cdot G. \tag{2}$$

   Then, A verifies the validation of the equation $H(a \cdot G, SK_A) = H(X, SK_B)$. If it holds, A authenticates B and sends $H(Y_1, SK_A)$ to B.
4. B verifies the validation of the equation $H(b \cdot G, SK_B) = H(Y_1, SK_A)$. If it holds, B authenticates A.

After the Step 4, A and B are now convinced the common secret session key $K = H(SK_A) = H(SK_B)$.

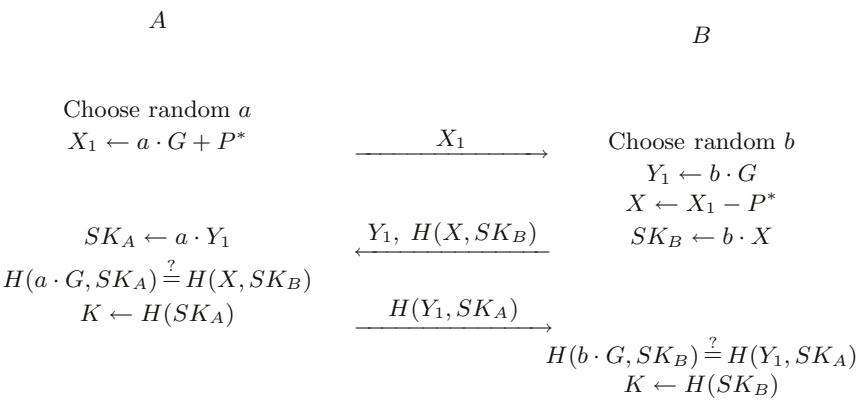


Fig. 2. Proposed ESAKA protocol

4 Security Analysis and Performance Comparison

This section discusses the security and efficiency of the proposed ESAKA protocol.

4.1 Security Analysis

This subsection provides the security analysis of the proposed ESAKA protocol. First, we define the security terms needed for security analysis of the proposed ESAKA protocol as follows:

Definition 1. *A weak secret (password) is a value of low entropy $W(k)$, which can be guessed in polynomial time.*

Definition 2. *The elliptic curve discrete logarithm problem ($ECDLP$) is as follows: given a public key point $Q_i = x_i \cdot G$, it is hard to compute secret key x_i.*

Definition 3. *The elliptic curve Diffie-Hellman problem ($ECDHP$) is as follows given point elements $a \cdot G$ and $b \cdot G$, it is hard to find $a \cdot b \cdot G$.*

Definition 4. *A secure one-way hash function $y = H(x)$ is one where given x to compute y is easy and given y to compute x is hard.*

Here, seven security properties: replay attack, password guessing attack, man-in-middle attack, modification attack, known-key security, session key security, and perfect forward secrecy, must be considered for the proposed ESAKA protocol. Under the above definitions, the following theorems are used to analyze seven security properties in the proposed protocol.

Theorem 1. *ESAKA protocol can resist the replay attack.*

Proof: Attacker C intercepts $X_1 = a \cdot G + P^*$ from A in Step 1 and uses it to impersonate A. However, C cannot compute a correct $H(Y_1, SK_A)$ and deliver it to B unless he/she can correctly guess password P to obtain Y_1 and guess the right b, and then C must face the ECDLP. On the other hand, suppose C intercepts Y_1, $H(X, SK_B)$ from B in Step 2 and uses it to impersonate B. For the same reason, if C cannot gain the correct a, A will find out that $H(a \cdot G, SK_A)$ is not equivalent to $H(X, SK_B)$, and then A will not send $H(Y_1, SK_A)$ back to C.

Theorem 2. *ESAKA protocol can resist the password guessing attack.*

Proof: An on-line password guessing attack cannot succeed since B can choose appropriate trail intervals. On the other hand, in an off-line password guessing attack, C can try to find out a weak password by repeatedly guessing possible passwords and verifying the correctness of the guesses based on information obtained in an off-line manner. In our protocol, C can gain the knowledge of $X_1 = a \cdot G + P^*$, Y_1, $H(X, SK_B)$ and $H(Y_1, SK_A)$ in Steps 1, 2, and 3, respectively. Assume that C wants to impersonate A. He/she first guesses password P' and then finds $X_1^* = X_1 - P'$ and Y_1. However, C has to break the ECDLP and ECDHP to find the keying material $SK_A = SK_B$ to verify his/her guess. C cannot gain the session key without X_1^* and the keying material SK_A, SK_B.

Theorem 3. *ESAKA protocol can resist the man-in-middle attack.*

Proof: A mutual password between A and B is used to prevent the man-in-middle attack. The illegal attacker C cannot pretend to be A or B to authenticate the other since he/she does not own the mutual password.

Theorem 4. *ESAKA protocol can resist the modification attack.*

Proof: An attacker may modify the messages X_1, Y_1, $H(X, SK_B)$, and $H(Y_1, SK_A)$ being transmitted over an insecure network. However, although the attacker forges them, ESAKA protocol can detect this attack, because it can verify not only the equality of SK_A and SK_B computed by each party, but also the correctness of X_1, Y_1 transmitted between two parties through validating $H(X, SK_B)$ and $H(Y_1, SK_A)$ in the protocol.

Theorem 5. *ESAKA protocol provides known-key security.*

Proof: Known-key security means that each run of a key agreement protocol between two entities A and B should produce unique secret keys; such keys are called session keys. Knowing a session key K and the random values a and b are of no use for computing the other session keys $H(a' \cdot b' \cdot G)$, since without knowing a' and b' it is impossible to compute the session key K.

Theorem 6. *ESAKA protocol provides session key security.*

Proof: Session key security means that at the end of the key exchange, the session key is not known by anyone but A and B. The session key $H(a \cdot b \cdot G)$ is not known by anyone but A and B since the random value a and b are protected by the ECDHP and the secure one-way hash function. None of this session key $K = H(a \cdot b \cdot G)$ is known to anybody but A and B.

Theorem 7. *ESAKA protocol provides perfect forward secrecy.*

Proof: Perfect forward secrecy means that if long-term private keys of one or more entities are compromised, the secrecy of previous session keys established by honest entities is not affected. If the user's password itself is compromised, it does not allow an attacker to determine the session key K for past sessions and decrypt them, since the attacker is still faced with the ECDHP. Therefore, the ESAKA protocol satisfies the property of perfect forward secrecy.

4.2 Performance Comparison

The computation costs of the proposed ESAKA protocol and previous SAKA-like protocols are summarized in Table 1. The elliptic curve discrete logarithm problem (ECDLP) with an order of 160 bit prime offers approximately the same level of security as the discrete logarithm problem(DLP) with 1024 bit modulus [2].

The ESAKA protocol requires four multiplications of a number and a point on the elliptic curve and six hash operations during the protocol. Therefore, the ESAKA protocol has a low computational load. In terms of network resource efficiency and network delay, it is advantageous to have as few communication rounds as possible. Therefore, the number of messages to be exchanged between communication parties should be kept to a minimum.

The ESAKA protocol requires three passes to perform a mutual authentication and key agreement. Therefore, the ESAKA protocol has a minimum number of message exchanges. The protocol message should be as short as possible. The ESAKA protocol requires four messages during the protocol. Among these four messages, two are exponentiation bits and two are hash output bits. Therefore, the ESAKA protocol uses a minimum communication bandwidth.

Table 1. Computation costs of ESAKA and Previous SAKA-like protocols

	SAKA [5]	T-SAKA [6]	KW-SAKA [8]	KHHL-SAKA [12]	ESAKA
# of steps	4	4	4	3	3
# of random numbers	2	2	2	2	2
# of exponentiations	10	8	9	8	4
# of hash functions	0	0	0	0	6
security	DLP	DLP	DLP	DLP	ECDLP

We compared ESAKA with other password-based key agreement protocols submitted to IEEE P1363.2 (Password-based Techniques) [15, 16] . Table 2 com-

pares various password-based protocols based on an asymmetric model. To compare the computational workload, we considered the number of exponentiations that consume the most execution time. In table 2, we use this counting method for a number of exponentiations.

AMP and SRP are four-pass protocols for password-based authenticated key exchange, but B-SPEKE and PAK-Z are three-pass protocols. AMP requires the smallest exponentiations and PAK-Z requires the smallest computational passes among the previously proposed protocols.

By contrast, ESAKA implements a three-pass protocol. In ESAKA, each party performs approximately two exponentiations and the exchanged data size is only $2|q| + 2|k|$. Therefore, as in Table 2, we can see that ESAKA has the smallest computational and communicational workloads.

Table 2. Computation costs of ESAKA and Various password-based protocols

	B-SPEKE	SRP	AMP	PAK-Z	ESAKA
# of steps	3	4	4	3	3
# of client's exponentiations	3	3	2	4	2
# of server's exponentiations	4	3	2.4	2	2
# of total exponentiations	7	6	4.4	6	4

5 Conclusions

The SAKA-like protocols are widely used in research for key agreement. However, previous SAKA-like protocols have some weakness. Therefore, several articles have been proposed to continuously enhance SAKA-like protocols. The current paper demonstrated the vulnerability of Kim et al.'s SAKA-like protocol to off-line password guessing attacks and the integrity violence of the session key from illegal modification. Then, to resolve such problems, we presented an ESAKA protocol based on the elliptic curve discrete logarithm problem (ECDLP). The proposed ESAKA protocol resists those attacks, while also providing more security and efficiency which can be executed faster than other previously proposed password-based protocols.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments to improve our manuscript. This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment).

References

1. Diffie, W., Hellman, M.: New Directions in Cryptography. IEEE Transaction on Information Theory. Vol. IT-22. No. 6. (1976) 644-654
2. Schneier, B.: Applied Cryptography-Protocols. Algorithms and Source Code in C. 2nd edi. John Wiley & Sons Inc. (1995)
3. Diffie, W., Van Oorschot, P.C., Wiener, M.J.: Authentication and Authenticated Key Exchanges. Design, Codes and Cryptography. Vol. 2. (1992) 107-125
4. Bellovin, S., Merritt, M.: Encrypted Key Exchange: Password-based Protocols Secure Against Dictionary Attacks. Proc. of IEEE Conf. on Research in Security and Privacy. (1992) 72-84
5. Seo, D.H., Sweeney, P.: Simple Authenticated Key Agreement Algorithm. Electronics Letters. Vol. 35. No. 13. (1999) 1073-1074
6. Tseng, Y.M.: Weakness in Simple Authenticated Key Agreement Protocol. Electronics Letters. Vol. 36. No. 1. (2000) 48-49
7. Lin, I.C., Chang, C.C., Hwang, M.S.: Security Enhancement for the Simple Authentication Key Agreement Algorithm. Proceedings of the 24th Annual International Computer Software and Application Conference. (2000) 113-115
8. Ku, W.C., Wang, S.D.: Cryptanalysis of Modified Authenticated Key Agreement Protocol. Electronics Letters. Vol. 36. No. 21. (2000) 1770-1771
9. Hsu, C.L., Wu, T.S., Wu, T.C., Mitchell, C.: Improvement of Modified Authenticated Key Agreement Protocol. Applied Mathematics and Computation. Vol. 142. No. 2-3. (2003) 305-308
10. Lee, N.Y., Lee, M.F.: Further Improvement on the Modified Authenticated Key Agreement Scheme. Applied Mathematics and Computation. Vol. 157. No. 3. (2004) 729-733
11. Ryu, E.K., Kim, K.W., Yoo, K.Y.: A Promising Key Agreement Protocol. ISAAC 2003. LNCS 2906. (2003) 655-662
12. Lee, S.W., Kim, H.S., Yoo, K.Y.: Improvement of Lee and Lee's Authenticated Key Agreement Scheme. Applied Mathematics and Computation. Vol. 162. No. 3. (2005) 1049-1053
13. Lee, S.W., Kim, H.S., Yoo, K.Y.: Improvement of HWWM-Authenticated Key Agreement Protocol. Applied Mathematics and Computation. Vol. 162. No. 3. (2005) 1315-1320
14. Kim, Y.S., Huh, E.N., Hwang, J.H., Lee, B.W.: An Efficient Key Agreement Protocol for Secure Authentication. ICCSA 2004. LNCS 3043. (2004) 746-754
15. Hwang, Y.H., Yum, D.H., Lee, P.J.: EPA: An Efiicient Password-Based Protocol for Authenticated Key Exchange. ACISP 2003. LNCS 2727. (2003) 452-463
16. The latest draft of IEEE P1363.2.: Standard Specifications for Password-Based Public Key Cryptography Techniques. Draft D19. December 17. (2004) http://grouper.ieee.org/groups/1363/

A Quadratic Lower Bound for Rocchio's Similarity-Based Relevance Feedback Algorithm

Zhixiang Chen[1] and Bin Fu[2,3]

[1] Department of Computer Science, University of Texas-Pan American
Edinburg, TX 78541, USA
chen@cs.panam.edu
[2] Department of Computer Science, University of New Orleans
New Orleans, LA 70148, USA
fu@cs.uno.edu
[3] Research Institute for Children
200 Henry Clay Avenue, New Orleans, LA 80118, USA

Abstract. It is shown in [4] that Rocchio's similarity-based relevance feedback algorithm makes $\Omega(n)$ mistakes in searching for a collection of documents represented by a monotone disjunction of at most k relevant features (or terms) over the n-dimensional binary vector space $\{0,1\}^n$. In practice, Rocchio's algorithm often uses a fixed query updating factor and a fixed classification threshold. When this is the case, we strengthen the work in [4] in this paper and prove that Rocchio's algorithm makes $\Omega(k(n-k))$ mistakes in searching for the same collection of documents over the binary vector space $\{0,1\}^n$. A quadratic lower bound is obtained when k is proportional to n. An $O(k(n-k)^2)$ upper bound is also obtained.

1 Introduction

Research on relevance feedback in information retrieval has a long history and becomes a necessary part of our daily life due to the popularity of the Web. It is regarded as the most popular query reformation strategy [1, 10]. The central idea of relevance feedback is to improve search performance for a particular query by modifying the query step by step, based on the user's judgments of the relevance or irrelevance of some of the documents retrieved. In his popular textbook [13], van Vijsbergen describes the relevance feedback as a fixed error correction procedure and relates it to the linear separation problem. When the inner product similarity is used, relevance feedback is just a Perceptron-like learning algorithm [6]. Wong, Yao and Bollmann [14] studied the linear structure in information retrieval. They designed a very nice gradient descent procedure to compute the coefficients of a linear function and analyzed its performance. In [2, 3], multiplicative adaptive algorithms are devised for user preference retrieval with provable, efficient performance.

There are many different variants of relevance feedback in information retrieval. However, in this paper we only study Rocchio's similarity-based relevance

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 955–964, 2005.

feedback algorithm [5, 10]. In spite of its popularity in various applications, there is little rigorous analysis of its complexity as a learning algorithm in literature. As a first step towards formal analysis of Rocchio's similarity-based relevance feedback algorithm, the work in [4] establishes a linear lower bound on classification mistakes for the algorithm over the binary vector space $\{0,1\}^n$, when any of the four typical similarities (inner product, dice coefficient, cosine coefficient, and Jaccard coefficient) listed in [10] is used. The linear lower bound obtained in [4] is independent of the query updating factor and the classification threshold that are used by the algorithm. A number of challenging problems regarding further analysis of the algorithm remain open [4].

In practice, a fixed query updating factor and a fixed classification threshold are often used in Rocchio's similarity-based relevance feedback algorithm [1, 10]. Using a fixed query updating factor has many merits, such as simplicity and efficiency. As another example, the popular Winnow algorithm [7] uses a fixed updating factor. When this is the case, one shall naturally ask whether the linear lower bound obtained in [4] can be further strengthened? The main contribution of this paper is to give a positive answer to this question.

2 Rocchio's Similarity-Based Relevance Feedback Algorithm

Let R be the set of all real values, and let R^+ be the set of all non-negative real values. Let n be a positive integer. In the binary vector space model in information retrieval [10, 12], a collection of n features (or terms) $T_1, T_2, \ldots, T_n$ are used to represent documents and queries. Each document $\mathbf{d}$ is represented as a vector $v_{\mathbf{d}} = (d_1, d_2, \ldots, d_n)$ such that for any i, $1 \leq i \leq n$, the i-th component of $v_{\mathbf{d}}$ is one if the i-th feature T_i appears in $\mathbf{d}$ or zero otherwise. Each query $\mathbf{q}$ is represented by a vector $v_{\mathbf{q}} = (q_1, q_2, \ldots, q_n)$ such that for any i, $1 \leq i \leq n$, the i-th component of $v_{\mathbf{q}} \in R$ is a real value used to determine the relevance (or weight) of the i-th feature T_i. Because of the unique vector representations of documents and queries, for convenience we simply use $\mathbf{d}$ and $\mathbf{q}$ to stand for their vector representations $v_{\mathbf{d}}$ and $v_{\mathbf{q}}$, respectively. A similarity in general is a function m from $R^n \times R^n$ to R^+. A similarity m is used to determine the *relevance closeness* of documents to the search query and to rank documents according to such closeness. The following four typical similarities were listed in [10]: For any $\mathbf{q}, \mathbf{x} \in R^n$,

$$\textit{inner product}: m_1(\mathbf{q},\mathbf{x}) = \sum_{i=1}^{n} q_i x_i,$$

$$\textit{dice coefficient}: m_2(\mathbf{q},\mathbf{x}) = \frac{2m_1(\mathbf{q},\mathbf{x})}{m_1(\mathbf{q},\mathbf{q}) + m_1(\mathbf{x},\mathbf{x})},$$

$$\textit{cosine coefficient}: m_3(\mathbf{q},\mathbf{x}) = \frac{m_1(\mathbf{q},\mathbf{x})}{\sqrt{m_1(\mathbf{q},\mathbf{q})}\sqrt{m_1(\mathbf{x},\mathbf{x})}},$$

$$\textit{Jaccard coefficient}: m_4(\mathbf{q},\mathbf{x}) = \frac{m_1(\mathbf{q},\mathbf{x})}{m_1(\mathbf{q},\mathbf{q}) + m_1(\mathbf{x},\mathbf{x}) - m_1(\mathbf{q},\mathbf{x})}.$$

To make the above definitions valid for arbitrary $\mathbf{q}$ and $\mathbf{x}$, we define that the similarity between two zero vectors is zero, i.e.,

$$m_i(\mathbf{0}, \mathbf{0}) = 0, \ for\ 1 \leq i \leq 4.$$

Definition 1. Let m from $R^n \times R^n$ to R^+ be a similarity. A classifier with respect to m over the n-dimensional binary vector space $\{0,1\}^n$ is a triple $(\mathbf{q}, \psi, m)$, where $\mathbf{q} \in R^n$ is a query vector, and $\psi \in R$ is a threshold. The classifier $(\mathbf{q}, \psi, m)$ classifies any documents $\mathbf{d} \in \{0,1\}^n$ as relevant if $m(\mathbf{q}, \mathbf{d}) \geq \theta$ or irrelevant otherwise. The classifier $(\mathbf{q}, \psi, m)$ is called a *linear classifier* with respect to the similarity m, if m is a linear function from $R^n \times R^n$ to R^+.

Definition 2. An adaptive supervised learning algorithm A for learning a target classifier $(\mathbf{q}, \psi, m)$ over the n-dimensional binary vector space $\{0,1\}^n$ from examples is a game played between the algorithm A and the user in a step by step fashion, where the query vector $\mathbf{q}$ and the threshold θ are unknown to the algorithm A, but the similarity m is. At any step $t \geq 1$, A gives a classifier $(\mathbf{q}_t, \theta_t, m)$ as a hypothesis to the target classifier to the user, where $\mathbf{q}_t \in R^n$ and $\theta_t \in R$. If the hypothesis is equivalent to the target, then the user says "yes" to conclude the learning process. Otherwise, the user presents an example $\mathbf{x_t} \in \{0,1\}^n$ such that the target classifier and the hypothesis classifier differ at $\mathbf{x_t}$. In this case, we say that the algorithm A makes a mistake. At step $t+1$, the algorithm A constructs a new hypothetical classifier $(\mathbf{q}_{t+1}, \theta_{t+1}, m)$ to the user based on the received examples $\mathbf{x_1}, \ldots, \mathbf{x_t}$. The learning complexity (or the mistake bound) of the algorithm A is in the worst case the maximum number of examples that it may receive from the user in order to learn some classifier.

Definition 3. Rocchio's similarity-based relevance feedback algorithm is an adaptive supervised learning algorithm for learning any classifier $(\mathbf{q}, \theta, m)$ over the n-dimensional binary vector space $\{0,1\}^n$ from examples. Let $\mathbf{q}_1$ be the initial query vector. At any step $t \geq 1$, the algorithm presents a classifier $(\mathbf{q}_t, \theta_t, m)$ as its hypothesis to the target classifier to the user, where $\theta_t \in R$ is the threshold, and the query vector $\mathbf{q}_t$ is modified as follows. Assume that at the beginning of step t the algorithm has received a sequence of examples $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$, then the algorithm uses the following modified query vector $\mathbf{q}_t$ for its next classification:

$$\mathbf{q}_t = \alpha_{t_0}\mathbf{q}_1 + \sum_{j=1}^{t-1} \alpha_{t_j}\mathbf{x}_j, \tag{1}$$

where $\alpha_{t_j} \in R$, for $j = 0, \ldots, t-1$, are called query updating factors.

In particular, when a fixed query updating factor $\alpha > 0$ and a fixed classification threshold θ are used, at any step $t \geq 1$, Rocchio's algorithm uses $(\mathbf{q}_t, \theta, m)$ as its hypothesis to the target classifier, and the query vector $\mathbf{q}_t$ is modified as

$$\mathbf{q}_t = \mathbf{q}_1 + \sum_{j=1}^{t-1} \alpha(y_j^* - y_j)\mathbf{x}_j, \tag{2}$$

where y_j^* is the binary classification value of the target classifier $(\mathbf{q}, \psi, m)$ on $\mathbf{x}_j$, and y_j is the binary classification value of the hypothesis classifier $(\mathbf{q}_t, \theta, m)$ on $\mathbf{x}_j$.

Please note that our definition above is a generalized version of Rocchio's original algorithm.

3 When the Classification Threshold Is Zero

We will use the sets of documents represented by monotone disjunctions of relevant features to study the mistake bounds of Rocchio's algorithm. The efficient learnability of monotone disjunctions of relevant features (or attributes) has been extensively studied in machine learning (for example, [7]). Although very simple in format, monotone disjunctions are very common ways of expressing search queries, especially in the case of Web search. All existing popular search engines support disjunctions of keywords as search query formations.

Proposition 1. *Let* $\mathbf{q}_{t+1}$ *be the query vector of Rocchio's algorithm at step* $t+1 \geq 2$. *Then, for any* s, $2 \leq s \leq 4$, *and any* $\mathbf{x} \in \{0,1\}^n$, *we have*

$$m_1(\mathbf{q}_{t+1}, \mathbf{x}) \geq 0 \Longleftrightarrow m_1(\sum_{j=1}^{t}(y_j^* - y_j)\mathbf{x}_j, \mathbf{x}) \geq 0$$

$$\Longleftrightarrow m_s(\sum_{j=1}^{t}(y_j^* - y_j)\mathbf{x}_j, \mathbf{x}) \geq 0, \text{ if } \mathbf{q}_1 = \mathbf{0},$$

$$m_1(\mathbf{q}_{t+1}, \mathbf{x}_j) \geq 0 \Longleftrightarrow m_s(\mathbf{q}_{t+1}, \mathbf{x}_j) \geq 0, \text{ if } \mathbf{q}_1 \neq \mathbf{0}.$$

Proof. It follows from expression (2) in Definition 3 and the fact that the enumerator of m_s is m_1 and the denominator of m_s is positive. □

Theorem 1. *Rocchio's similarity-based relevance feedback algorithm with similarity* m_s, $1 \leq s \leq 4$, *makes at least* $3k(n-k)$ *mistakes in searching for the collection of documents represented by a monotone disjunction of* k *relevant features over the binary vector space* $\{0,1\}^n$, *when the initial query vector* $\mathbf{q}_1 = \mathbf{0}$, *and the algorithm uses a fixed query updating factor* $\alpha > 0$ *and a fixed classification threshold* $\theta = 0$.

Proof. By Proposition 1, we only need to consider the similarity m_1 and the query updating factor $\alpha = 1$. Without loss of generality, let us work on the monotone disjunction of k relevant features

$$F_k = X_1 \vee \cdots \vee X_k. \tag{3}$$

The idea is that, for $j = 1, \ldots, k$, we construct a sequence of examples that allow the algorithm to focus on the learning of the relevant feature X_j exclusively.

The algorithm gains no information for the other relevant features, hence it is unable to update the components of the query vector corresponding to the relevant features other than X_j.

We consider the phrase for learning the relevant feature X_1. This phrase is divided into two steps: The preprocessing step and the learning step. We start with the preprocessing step. We construct examples $\mathbf{x}_t$, $t = 1, \ldots, n-k$, such that $x_{t,k+t} = 1$ and all its other components are 0. $\mathbf{x}_1$ does not satisfy F_k. But $m_1(\mathbf{q}_1, \mathbf{x}_1) = 0$, classifying x_1 as relevant. Hence, the algorithm makes one mistake, and the query vector $\mathbf{q}_2$ is updated as $q_{2,k+1} = -1$ and $q_{2,j} = q_{1,j}$ for $1 \leq j \leq n$ and $j \neq k+1$. Similarly, for $t \geq 2$, the algorithm makes a mistake on $\mathbf{x}_t$ and sets $q_{t+1,k+t} = -1$ and $q_{t+1,j} = q_{t,j}$ for $1 \leq j \leq n$ and $j \neq k+t$. At the end of this step, the query vector $\mathbf{q}_{n-k+1}$ is updated as $q_{n-k+1,j} = 0$ for $j = 1, \ldots, k$, and $q_{n-k+1,k+j} = -1$ for $j = 1, \ldots, n-k$.

We then begin the learning step. We construct examples $\mathbf{y}_t$ for $t = 1, \ldots, 2(n-k)$. We set $y_{t,1} = 1$ if t is odd and $y_{t,1} = 0$ if t is even. For all t, we set $y_{t,l} = 0$ for $2 \leq l \leq k$, and $y_{t,k+l} = 1$ for $1 \leq l \leq n-k$. Obviously, $\mathbf{y}_t$ satisfies F_k if t is odd, and it does not if t is even. For $\mathbf{y}_1$, $m_1(\mathbf{q}_{n-k+1}, \mathbf{y}_1) = -(n-k) < 0$. This implies that the algorithm makes a mistake, and the query vector $\mathbf{q}_{n-k+2}$ is updated as $q_{n-k+2,1} = 1$, $q_{n-k+2,l} = 0$ for $l = 2, \ldots, k$, and $q_{n-k+2,k+l} = 0$ for $l = 1, \ldots, n-k$. For $\mathbf{y}_2$, $m_1(\mathbf{q}_{n-k+2}, \mathbf{x}_{n-k+2}) = 0$, implying that the algorithm makes another mistake, and the query vector $\mathbf{q}_{n-k+3}$ is updated as $q_{n-k+3,1} = 1$, $q_{n-k+3,l} = 0$ for $l = 2, \ldots, k$, and $q_{n-k+3,k+l} = -1$ for $l = 1, \ldots, n-k$. It follows from the similar analysis that the algorithm makes one mistake for each $\mathbf{y}_t$, and the query vector has the following property: For $t = 1, 3, \ldots, 2(n-k)-1$, $q_{n-k+t+1,1} = (t+1)/2$, $q_{n-k+t+1,l} = 0$ for $l = 2, \ldots, k$, and $q_{n-k+t+1,k+l} = 0$ for $l = 1, \ldots, n-k$; for $t = 2, 4, \ldots, 2(n-k)$, $q_{n-k+t+1,1} = t/2$, $q_{n-k+t+1,l} = 0$ for $l = 2, \ldots, k$, and $q_{n-k+t+1,n+l} = -1$ for $l = 1, \ldots, n-k$.

In summary, both the preprocessing step and the learning step for the relevant feature X_1 force the algorithm to make at least $3(n-k)$ mistakes. Moreover, at the end of this phrase, the query vector $\mathbf{q}_{3(n-k)+1}$ is updated as $q_{3(n-k)+1,1} = n-k$, $q_{3(n-k)+1,l} = 0$ for $l = 2, \ldots, k$, and $q_{3(n-k)+1,k+l} = 0$ for $l = 1, \ldots, n-k$. By simple induction, we can utilize the similar preprocessing and learning steps to force the algorithm to make $3(n-k)$ mistakes to learn each of the remaining $k-1$ relevant features. Therefore, the algorithm makes at least $3k(n-k)$ mistakes in searching for the collection of documents represented by F_k. □

Theorem 2. *Rocchio's similarity-based relevance feedback algorithm with similarity m_s, $1 \leq s \leq 4$, makes $\Omega(k(n-k))$ mistakes in searching for documents represented by a monotone disjunction of k relevant features over the binary vector space $\{0,1\}^n$, when the initial query vector $\mathbf{q}_1 \in \{0,1\}^n$ is not $\mathbf{0}$, the query updating factor α is a constant and the classification threshold $\theta = 0$.*

Proof. By Proposition 1, we only need to consider the similarity m_1. Again, we work on the monotone disjunction F_k, defined in expression (3), of k relevant features. The idea of proof is similar to that of Theorem 1, i.e., for each relevant feature, we construct examples that force the algorithm to focus on the learning of that feature exclusively.

We shall address two cases, the initial query vector $\mathbf{q}_1$ has either at least k zero components or less than k zero components.

Case 1: In this case, we assume without loss of generality that $q_{1,i} = 0$ for $i = 1, \ldots, k$, $q_{1,k+j} = 0$ for $j = 1, \ldots, m$, and $q_{1,k+m+j} = 1$ for $j = 1, \ldots, n-k-m$, where $0 \leq m \leq n-k$.

We start with the initial phrase to set the $(k+j)$-th component of the query vector to $-\alpha$ for $j = 1, \ldots, m$. We construct examples $\mathbf{x}_t$ for $t = 1, \ldots, m$ such that $x_{t,k+t} = 1$ and all its other components are 0. $\mathbf{x}_t$ does not satisfy the given monotone disjunction F_k. By simple induction, $m_1(\mathbf{q}_t, \mathbf{x}_t) = 0$, implying that $\mathbf{x}_t$ is classified by the algorithm as relevant. Hence, the algorithm makes one mistake on $\mathbf{x}_t$, and the $(k+t)$-th component of the query vector $\mathbf{q}_{t+1}$ is updated as $q_{t+1,k+t} = -\alpha$ and all the other components are the same as those in $\mathbf{q}_t$. At the end, the query vector $\mathbf{q}_{m+1}$ is updated as $q_{m+1,j} = 0$ for $1 \leq j \leq k$ or $m+k+1 \leq j \leq n$, and $q_{m+1,k+j} = -\alpha$ for $j = 1, \ldots, m$. To simplify notation, we let $\mathbf{A}_1 = \mathbf{q}_{m+1}$.

The second phrase is to manipulate the $n-k-m$ many components of $\mathbf{A}_1$ with value 1. Let T be the smallest integer satisfying $T > 1/\alpha$. For each j with $k+m+1 \leq j \leq n$, we construct examples $\mathbf{y}_t$ for $t = 1, \ldots, T$ such that $y_{t,j} = 1$ and all the other components are 0. $\mathbf{y}_t$ does not satisfy the given monotone disjunction F_k. By simple induction, $m_1(\mathbf{A}_t, \mathbf{y}_t) = 1-(t-1)\alpha$, which is greater than or equal to zero because of the choice of T. This implies that the algorithm classifies $\mathbf{y}_t$ as relevant. Hence, the algorithm makes one mistake on $\mathbf{y}_t$, and the $(k+m+j)$-th component of the query vector $\mathbf{A}_{t+1}$ is updated as $A_{t+1,j} = 1 - t\alpha$ and all the other components are the same as those in $\mathbf{A}_t$. At the end of this phrase, the algorithm makes at least $T(n-k-m)$ mistakes, and the query vector $\mathbf{A}_{T(n-k-m)+1}$ is updated as $A_{T(n-k-m)+1,l} = 0$ for $1 \leq l \leq k$, $A_{T(n-k-m)+1,k+l} = -\alpha$ for $j = 1, \ldots, m$, and $A_{T(n-k-m)+1,k+m+l} = 1 - T\alpha$ for $l = 1, \ldots, n-k-m$. To simplify notation, we let $\mathbf{B}_1 = \mathbf{A}_{T(n-k-m)+1}$

The third phrase is for learning the relevant feature X_1. We construct examples $\mathbf{z}_t$ for $t = 1, \ldots, 2(m+(n-k-m)(T-\frac{1}{\alpha}))$. We set $z_{t,1} = 1$ if t is odd, and $z_{t,1} = 0$ if t is even. For all t, we set $z_{t,l} = 0$ for $2 \leq l \leq k$, and $z_{t,k+l} = 1$ for $1 \leq l \leq n-k$. Obviously, $\mathbf{z}_t$ satisfies the given monotone disjunction F_k if t is odd, and it does not if t is even. By simple induction, when t is odd, we have $m_1(\mathbf{B}_t, \mathbf{z}_t) = (t-1)\alpha - m\alpha + (n-k-m)(1-T\alpha) < 0$; and when t is even, we have $m_1(\mathbf{B}_t, \mathbf{z}_t) = (n-k-m)(1-(T-1)\alpha - 1) \geq 0$. This means that the algorithm makes a mistake for each example $\mathbf{z}_t$, and the query vector is updated as follows: For $t = 1, 3, \ldots, 2(m+(n-k-m)(T-\frac{1}{\alpha})) - 1$, $B_{t+1,1} = \frac{(t+1)\alpha}{2}$, $B_{t+1,l} = 0$ for $l = 2, \ldots, k+m$, and $B_{t+1,l} = 1-(T-1)\alpha$ for $l = k+m+1, \ldots, n$; for $t = 2, 4, \ldots, 2(m+(n-k-m)(T-\frac{1}{\alpha}))$, $B_{t+1,1} = \frac{t\alpha}{2}$, $B_{t+1,l} = 0$ for $j = 2, \ldots, k$, $B_{t+1,l} = -\alpha$ for $l = k+1, \ldots, k+m$, and $B_{t+1,l} = 1 - T\alpha$ for $l = k+m+1, \ldots, n$.

The above analysis implies that the algorithm makes $2(m+(n-k-m)(T-\frac{1}{\alpha}))$ mistakes in the phrase of learning the relevant feature X_1. It is interesting to notice that the query vector at the beginning and at the end of this phrase remains the same, except its first component. Precisely, we have

$$B_{1,1} = 0, \text{ but } B_{2(m+(n-k-m)(T-\frac{1}{\alpha}))+1,1} = m + (n-k-m)(T - \frac{1}{\alpha}),$$
$$B_{1,j} = B_{2(m+(n-k-m)(T-\frac{1}{\alpha}))+1,j} = 0, j = 2, \ldots, k,$$
$$B_{1,j} = B_{2(m+(n-k-m)(T-\frac{1}{\alpha}))+1,j} = -\alpha, j = k+1, \ldots, k+m, \text{ and}$$
$$B_{1,j} = B_{2(m+(n-k-m)(T-\frac{1}{\alpha}))+1,j} = 1 - T\alpha, j = k+m+1, \ldots, n.$$

We can apply a similar phrase to the phrase for learning the relevant feature X_1 for each of the remaining relevant features X_i, $2 \leq i \leq k$. That is, we can construct a new example $\mathbf{z}'_t$ from each $\mathbf{z}_t$. The new example $\mathbf{z}'_t$ is obtained via changing the i-th component of $\mathbf{z}_t$ to 1 if t is odd, and to 0 if t is even, the j-th component to zero for $1 \leq j \leq k$ and $j \neq i$, and letting the other $n-k$ components remain the same as in $\mathbf{z}_t$. By simple induction, the algorithm will make one mistake on each $\mathbf{z}'_t$, the query vector satisfies the similar invariant property as shown in the above expressions. Therefore, the algorithm makes at least $2k(m + (n-k-m)(T - \frac{1}{\alpha}))$ mistakes in all the k phrases of learning the relevant feature X_i for $1 \leq i \leq k$. Combining with the first two phrases, the algorithm makes in total at least $m+(n-k-m)T+2k(m+(n-k-m)(T-\frac{1}{\alpha})) = \Omega(k(n-k))$ mistakes.

Case 2: In the second case, we assume without loss of generality that $q_{1,i} = 0$ for $i = 1, \ldots, m'$ where $0 \leq m' < k$, $q_{1,i} = 1$ for $i = m'+1, \ldots, k$, and $q_{1,k+j} = 1$ for $j = 1, \ldots, n-k$.

In this case, we do not need the initial phrase in Case 1. For consistence with Case 1, we let $\mathbf{A}_1 = \mathbf{q}_1$. We first follow the second phrase in Case 1 to manipulate, for each j, $1 \leq j \leq n-k$, the 1-component $A_{1,k+j}$ by constructing examples $\mathbf{y}_t$ for $t = 1, \ldots, T$. Like in the second phrase in Case 1, each of such examples forces the algorithm to make a mistake, and to update $A_{t+1,j}$ to $1 - t\alpha$. At the end of this phrase, the algorithm makes $(n-k)T$ mistakes, and the query vector $\mathbf{A}_{(n-k)T+1}$, denoted as $\mathbf{B}_1$ to simply notation, becomes $B_{1,l} = 0$ for $1 \leq l \leq m'$, $B_{1,l} = 1$ for $m'+1 \leq l \leq k$, and $B_{1,k+l} = 1 - T\alpha$ for $l = 1, \ldots, n-k$.

For i, $1 \leq i \leq m'$, we follow the same phrase as that in Case 1 for learning the relevant feature X_i by constructing examples $\mathbf{z}_t$ for $t = 1, \ldots, 2(n-k)(T - \frac{1}{\alpha})$. The only difference is that here we have $m = 0$. Like in that phrase in Case 1, these examples force the algorithm to make $2(n-k)(T - \frac{1}{\alpha})$ mistakes, and the query vector maintains the similar invariant property as in Case 1.

For i, $m'+1 \leq i \leq k$, we again follow the same phrase as that in Case 1 for learning the relevant feature X_i by constructing examples $\mathbf{z}_t$ for $t = 1, \ldots, 2((n-k)(T - \frac{1}{\alpha}) - \frac{1}{\alpha})$. The two differences that we have here are $m = 0$ and $B_{1,i} = 1$. The fact of $m = 0$ will not change the process. But the fact of $B_{1,i} = 1$ will change the number of examples to $2((n-k)(T - \frac{1}{\alpha}) - \frac{1}{\alpha})$, because the i-th component of the query vector becomes $1 + \frac{(t+1)\alpha}{2}$ for odd t, and $1 + \frac{t\alpha}{2}$ for even t. Again, these examples force the algorithm to make $2((n-k)(T - \frac{1}{\alpha}) - \frac{1}{\alpha})$ mistakes, and the similar invariant property is maintained for the query vector. Putting the above analysis together, an $\Omega(k(n-k))$ lower bound is obtained in Case 2. □

4 When the Classification Threshold Is Not Zero

One can introduce an auxiliary feature variable to deal with the threshold so that a linear classifier over the n-dimensional vector space with a non-zero threshold is equivalent to a linear classifier over the $(n+1)$-dimensional vector space with a zero threshold. Thus, it is tempting to use Theorems 1 and 2 to derive lower bounds for Rocchio's algorithm with non-zero classification threshold over the $(n+1)$-dimensional binary vector space. However, one shall notice that the auxiliary feature variable will always maintain a value 1 in any example given to the algorithm and the threshold may have arbitrary values other than just 1 or 0, therefore the proofs of these two theorems cannot be applied here.

Theorem 3. *Rocchio's similarity-based relevance feedback algorithm with similarity m_1 makes $\Omega(k(n-k))$ mistakes in searching for the collection of documents represented by a monotone disjunction of k relevant features over the binary vector space $\{0,1\}^n$, when the query updating factor α is a constant, the initial query vector $\mathbf{q}_1 = \mathbf{0}$, and the classification threshold $\theta \neq 0$.*

Proof. As in the previous section, let us consider the monotone disjunction F_k defined in expression (3), and for each i, $1 \leq i \leq k$, we construct examples that allow the algorithm to focus on the learning of the relevant feature X_i exclusively. We shall analyze several cases for the classification threshold θ.

Case 1: $\theta \geq (n-k)\alpha$. For each relevant feature X_i, $1 \leq i \leq k$, we construct a sequence of examples $\mathbf{x}_t, t = 1, \ldots, (n-k+1)$, such that $x_{t,i} = 1$ and $x_{t,j} = 0$ for $1 \leq j \leq n$ and $j \neq i$. Note that $\mathbf{x}_t$ satisfies F_k. Since $\mathbf{q}_1 = \mathbf{0}$, by simple induction, we have $m_1(\mathbf{q}_{(i-1)(n-k+1)+t}, \mathbf{x}_t) = (t-1)\alpha < \theta$. This implies that the algorithm makes a mistake on $\mathbf{x}_t$, and the query vector is updated as $\mathbf{q}_{(i-1)(n-k+1)+t+1,i} = t\alpha$, $\mathbf{q}_{(i-1)(n-k+1)+t+1,j} = (n-k+1)\alpha$ for $1 \leq j \leq i-1$, and $\mathbf{q}_{(i-1)(n-k+1)+t+1,j} = 0$. At the end, the algorithm makes at least $k(n-k+1)$ mistakes in learning all the k relevant features in F_k.

Case 2: $0 < \theta < (n-k)\alpha$. We choose the integer T such that $(T-1)\alpha < \theta \leq T\alpha$, $1 \leq T \leq (n-k)$. Let $N = \lfloor \frac{n-k}{T} \rfloor$. We first construct an example $\mathbf{x}_1$ such that $x_{1,i} = 1$ for $1 \leq i \leq k$, $x_{1,k+j} = 1, j = 1, \ldots, NT$, and the remaining components are zero. This example satisfies F_k, but $m_1(\mathbf{q}_1, \mathbf{x}_1) = 0$ since $\mathbf{q}_1 = \mathbf{0}$. Hence, the algorithm makes a mistake, and the query vector is updated as $q_{2,i} = \alpha$ for $1 \leq i \leq k$, $q_{2,k+j} = \alpha$ for $j = 1, \ldots, NT$, and all the other components remain 0. Now, for each relevant feature X_i, $1 \leq i \leq k$, we construct examples $\mathbf{x}_t$ for $t = 2, \ldots, T$ such that $x_{t,i} = 1$ and all its other components are zero. This example satisfies F_k, but $m_1(\mathbf{q}_t, \mathbf{x}_t) = (t-1)\alpha$, which is less than θ. Hence, the algorithm makes one mistakes on each $\mathbf{x}_t$, and the query vector is updated as $q_{t+1,i} = t\alpha$ and all the other components remain unchanged. At the end, the algorithm makes $k(T-1)+1$ mistakes, and $q_{k(T-1)+1,i} = T\alpha$, $q_{k(T-1)+1,k+j} = \alpha$ for $1 \leq j \leq NT$, and all the other components are zero.

Next, we construct examples $\mathbf{y}_t$ for $t = 1, \ldots, N-1$ such that $y_{t,k+T(t-1)+j} = 1, j = 1, \ldots, T$, and all its other components are zero. Each $\mathbf{y}_t$ does not satisfy

F_k, but $m_1(\mathbf{q}_{k(T-1)+t}, \mathbf{y}_t) = T\alpha \geq \theta$. This forces the algorithm to make one mistake on $\mathbf{y}_t$ and to set $q_{k(T-1)+t+1,k+T(t-1)+j} = 0$ for $j = 1, \ldots, T$, while keeping all the other components unchanged. For $t = N$, we set $y_{t,k+j} = 1$ for $j = 1, \ldots, NT$, and all the other components are zero. Again, this example does not satisfy F_k and forces the algorithm to make one mistake. The query vector is updated as $q_{k(T-1)+N+1,i} = T\alpha$, $q_{k(T-1)+N+1,j} = 0$ for $1 \leq j \leq k$ and $j \neq i$, $q_{k(T-1)+N+1,k+j} = -\alpha$, for $j = 1, \ldots, (N-1)T$, and $q_{k(T-1)+N+1,k+j} = 0$, for $j = (N-1)T+1, \ldots, NT$. We let $\mathbf{A}_1 = \mathbf{q}_{k(T-1)+N+1}$ to simplify notation.

We now consider the phrase to continue the learning of the relevant feature X_i, for $1 \leq i \leq k$. We construct examples $\mathbf{z}_t$ for $t = 1, \ldots, 2(N-1)T+1$ such that $z_{t,i} = 1$ if t is odd and $z_i = 0$ if t is even, $z_{t,k+j} = 1$ for $j = 1, \ldots, NT$, and all the other components are zero. Notice that $\mathbf{z}_t$ satisfies F_k if and only if t is odd. By simple induction, the algorithm makes one mistake on each $\mathbf{z}_t$, and the query vector is updated as $A_{t+1,i} = T\alpha + \frac{(t+1)\alpha}{2}$ for odd t, $A_{t+1,i} = T\alpha + \frac{t\alpha}{2}$ for even t, $A_{t+1,j} = 0$ for $1 \leq j \leq k$ and $j \neq i$ or $NT+1 \leq j \leq n$, $A_{t+1,k+j} = 0$ for odd t and $1 \leq j \leq (N-1)T$, $A_{t+1,k+j} = -\alpha$ for even t and $1 \leq j \leq (N-1)T$, $A_{t+1,k+j} = \alpha$ for odd t and $(N-1)T+1 \leq j \leq NT$, $A_{t+1,k+j} = 0$ for even t and $(N-1)T+1 \leq j \leq NT$. At the end of this phrase, the algorithm makes $2(N-1)T+1$ mistakes and, the query vector $\mathbf{A}_{2(N-1)T+1}$ maintains the following invariant property: all its components are the same as those in $\mathbf{A}_1$, except that the i-component is updated successfully to learn the relevant feature X_i. This invariant property allows us to follow the similar phrase for all the other relevant features to force the algorithm to make $2(N-1)T+1$ mistakes for each of such phrase. Therefore, the algorithm makes $2k(N-1)T+k$ mistakes during these phrases.

In summary, in this case the algorithm make at least $k(T-1)+N+1+2k(N-1)T+k = \Omega(k(n-k))$ mistakes, since $1 \leq T \leq (n-k)$ and $N = \lfloor \frac{n-k}{T} \rfloor$.

We omit the analysis for **Case 3** of $\theta < -(n-k)\alpha$ and **Case 4** of $-(n-k)\alpha < \theta < 0$ due to space limitation. □

Theorem 4. *Rocchio's similarity-based relevance feedback algorithm with similarity m_1 makes $\Omega(k(n-k))$ mistakes in searching for the collection of documents represented by a monotone disjunction of k relevant features over the binary vector space $\{0,1\}^n$, when the query updating factor α is a constant, the initial query vector is $\mathbf{q}_1 \in \{0,1\}^n$ is not $\mathbf{0}$, and the classification threshold $\theta \neq 0$.*

Proof Sketch. We follow the similar approach as in the proof for Theorem 3. But here we need to consider the additional value 1 that appear at the initial query vector when constructing examples forcing the algorithm to make mistakes. □

Remark 1. Theorems 3 and 4 can be extended to similarities m_s, $2 \leq s \leq 4$. The underlying analysis is, however, more complicated, and will be presented in the full version of the paper.

5 Conclusions

We devise the following upper bound for Rocchio's algorithm:

Theorem 5. *When a fixed updating factor and a zero classification threshold are used, Rocchio's algorithm with similarity m_1 makes at most $O(k(n-k)^2)$ mistakes in searching for the collection of documents represented by a monotone disjunction of k relevant features over the n-dimensional binary vector space.*

It is interesting to close the $(n-k)$-factor gap between the above upper bound and the lower bounds obtained in the previous two sections. Since Rocchio's algorithm uses linear classifiers as its hypotheses, it would be interesting to investigate the learning complexity of the algorithm in searching for documents represented by a linear classifier.

References

1. R. Baeza-Yates and B. Ribeiro-Neto B, *Modern Information Retrieval,* Addison-Wesley, 1999.
2. Z. Chen, Multiplicative adaptive algorithms for user preference retrieval, Proceedings of the Seventh Annual International Computing and Combinatorics Conference, LNCS 2108, pp. 540-549, Springer-Verlag, 2001.
3. Z. Chen, Multiplicative adaptive user preference retrieval and its applications to Web search, In: Zhang G, Kandel A, Lin T, and Yao Y, eds. *Computational Web Intelligence: Intelligent Technology for Web Applications*, pp. 303-328, World Scientific, 2004.
4. Z. Chen and B. Zhu, (2002) Some formal analysis of Rocchio's similarity-based relevance feedback algorithm, *Information Retrieval* **5**: 61-86, 2002. (The preliminary version of this paper appeared in *Proceedings of the Eleventh International Symposium on Algorithms and Computation (ISAAC'00)*, LNCS 1969, pp. 108-119, 2000.)
5. J. Rocchio, Relevance feedback in information retrieval, In: Salton G, ed. *The Smart Retrieval System - Experiments in Automatic Document Processing*, pp. 313-323, Prentice-Hall, Englewood Cliffs, NJ, 1971.
6. D. Lewis, Learning in intelligent information retrieval, Proceedings of the Eighth International Workshop on Machine Learning, pp. 235-239, 1991.
7. N. Littlestone, Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm, *Machine Learning*, **2**:285–318, 1988.
8. V. Raghavan and S. Wong, A critical analysis of the vector space model for information retrieval, *Journal of the American Society for Information Science*, **37**(5):279-287, 1986.
9. F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, **65**(6):386-407, 1958.
10. G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, 1989.
11. G. Salton and C. Buckley, Improving retrieval performance by relevance feedback, *Journal of the American Society for Information Science* **41**(4):288-297, 1990.
12. G. Salton, S. Wong and C. Yang, A vector space model for automatic indexing, *Comm. of ACM*, **18**(11):613-620, 1975.
13. C.J. van Vijsbergen, *Information Retrieval*. Butterworths, 1979.
14. S. Wong, Y. Yao and P. Bollmann, Linear structures in information retrieval, Proceedings of the 1988 ACM-SIGIR Conference on Information Retrieval, pp. 219-232, 1988.

The Money Changing Problem Revisited: Computing the Frobenius Number in Time $O(k\,a_1)^\star$

Sebastian Böcker and Zsuzsanna Lipták

AG Genominformatik, Technische Fakultät
Universität Bielefeld
PF 100 131
33501 Bielefeld, Germany
{boecker,zsuzsa}@CeBiTec.uni-bielefeld.de

Abstract. The Money Changing Problem (also known as Equality Constrained Integer Knapsack Problem) is as follows: Let $a_1 < a_2 < \cdots < a_k$ be fixed positive integers with $\gcd(a_1, \ldots, a_k) = 1$. Given some integer n, are there non-negative integers $x_1, \ldots, x_k$ such that $\sum_i a_i x_i = n$? The *Frobenius number* $g(a_1, \ldots, a_k)$ is the largest integer n that has no decomposition of the above form.
There exist algorithms that, for fixed k, compute the Frobenius number in time polynomial in $\log a_k$. For variable k, one can compute a residue table of a_1 words which, in turn, allows to determine the Frobenius number. The best known algorithm for computing the residue table has runtime $O(k\,a_1 \log a_1)$ using binary heaps, and $O(a_1(k + \log a_1))$ using Fibonacci heaps. In both cases, $O(a_1)$ extra memory in addition to the residue table is needed. Here, we present an intriguingly simple algorithm to compute the residue table in time $O(k\,a_1)$ and extra memory $O(1)$. In addition to computing the Frobenius number, we can use the residue table to solve the given instance of the Money Changing Problem in constant time, for any n.

1 Introduction

In the classical Money Changing Problem (MCP), we are given coins of k different values $a_1 < a_2 < \cdots < a_k$ with $\gcd(a_1, \ldots, a_k) = 1$. We want to know what change $n = \sum_i a_i x_i$ we can generate from these coins for non-negative integers x_i, assuming that we have an infinite supply of coins. Then, there exists an integer $g(a_1, \ldots, a_k)$ called the *Frobenius number* of $a_1, \ldots, a_k$, such that $g(a_1, \ldots, a_k)$ does not allow a decomposition of the above type, but all integers $n > g(a_1, \ldots, a_k)$ do. There has been considerable work on bounds for Frobenius numbers (see [16] for a survey) but here we concentrate on *exact* computations.

In 1884, Sylvester asked for the Frobenius number of $k = 2$ coins a_1, a_2, and Curran Sharp showed that $g(a_1, a_2) = a_1 a_2 - a_1 - a_2$ [17]. For three coins

* Supported by "Deutsche Forschungsgemeinschaft" (BO 1910/1-1 and 1-2) within the Computer Science Action Program

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 965–974, 2005.

a_1, a_2, a_3, Greenberg [9] provides a fast algorithm with runtime $O(\log a_1)$, and Davison [6] independently discovered a simple algorithm with runtime $O(\log a_2)$. Kannan [11] establishes algorithms that for *any fixed* k, compute the Frobenius number in time polynomial in $\log a_k$. For variable k, the runtime of such algorithms has a double exponential dependency on k, and is not competitive for $k \geq 5$. Also, it does not appear that Kannan's algorithms have actually been implemented. Computing the Frobenius number is NP-hard [15], so we cannot hope to find algorithms polynomial in k and $\log a_k$ simultaneously unless P = NP.

There has been a considerable amount of research on computing the exact Frobenius number if k is variable, see again [16] for a survey. Heap and Lynn [10] suggest an algorithm with runtime $O(a_k^3 \log g)$, and Wilf's "circle-of-light" algorithm [18] runs in $O(kg)$ time, where $g = O(a_1 a_k)$ is the Frobenius number of the problem. Until recently, the fastest algorithm to compute $g(a_1, \ldots, a_k)$ was due to Nijenhuis [14]: It is based on Dijkstra's method for computing shortest paths [7] using a priority queue. This algorithm has runtime $O(k\, a_1 \log a_1)$ using binary heaps, and $O(a_1(k + \log a_1))$ using Fibonacci heaps as the priority queue. To find the Frobenius number, the algorithm computes a data structure (called "residue table" in the following) of a_1 words, which in turn allows simple computation of $g(a_1, \ldots, a_k)$. Nijenhuis' algorithm requires $O(a_1)$ extra memory in addition to the residue table. Recently, Beihoffer et al. [3] developed algorithms that work faster in practice, but none obtains asymptotically better runtime bounds than Nijenhuis's algorithm, and all require extra memory linear in a_1.

Here, we present an intriguingly simple algorithm to compute the residue table – and hence, to find $g(a_1, \ldots, a_k)$ – in time $O(k\, a_1)$ with constant extra memory. In addition to the improved runtime bound, our algorithm also outperforms Nijenhuis' algorithm in practice.

Moreover, access to the residue table allows us to solve subsequent MCP *decision* problems on the same coin set in *constant* time: Here, the input is $a_1, \ldots, a_k, n$ and we ask the question "Is n decomposable?" The MCP decision problem, also known as Equality Constrained Integer Knapsack Problem, is also NP-hard [13]. In principle, one can solve MCP decision problems using generating functions, but the computational cost for coefficient expansion and evaluation are too high in applications [8, Chapter 7.3]. It is computer science folklore that the question can be answered in time $O(k\, n)$ using dynamic programming[1], but the linear runtime dependence on n is unfavorable. Aardal et al. [1, 2] use lattice basis reduction to find a decomposition of n. In Section 6 we show that computing the complete residue table using our algorithm is often faster than solving a single MCP decision instance with the method suggested by Aardal et al.

We show in [4] that a slightly modified version of the algorithm presented here allows its application for the analysis of mass spectrometry data: There, one is given a weighted alphabet (such as amino acids) and an input mass m, and one wants to find *all* decompositions of m. To this end, an "extended residue

[1] Wright [19] shows how to find a decomposition using a minimal number of coins in time $O(k\, n)$

table" of size $O(ka_1)$ is generated in runtime $O(ka_1)$, where a_1 is the smallest mass in the alphabet and k the size of the alphabet. This data structure allows for computation of all decompositions in time linear in the size of the output, and otherwise independent of the input mass m. Confer [4] for details.

2 Residue Classes and the Frobenius Number

For integers a and b, let "a mod b" denote the unique integer $p \in \{0, \ldots, b-1\}$ such that $p \equiv a \pmod{b}$ holds.

Let $a_1 < \cdots < a_k$ with $\gcd(a_1, \ldots, a_k) = 1$ be an instance of the Money Changing Problem. Brauer and Shockley [5] suggest to construct a *residue table* $(n_p)_{p=0,\ldots,a_1-1}$, where n_p is the smallest integer with $n_p \equiv p \pmod{a_1}$ that can be decomposed into a non-negative integer combination of $a_1, \ldots, a_k$. The n_p are well-defined: If n has a decomposition, so has $n + a_1$, and $n \equiv n + a_1 \pmod{a_1}$. Clearly, $\sum_i a_i x_i = n_p$ implies $x_1 = 0$ because otherwise, $n_p - a_1$ has a decomposition, too. If the n_p are known, then we can test in constant time whether some number n can be decomposed: Set $p = n \bmod a_1$, then n can be decomposed if and only if $n \geq n_p$.

Given the values n_p for $p = 0, \ldots, a_1 - 1$, we can compute the Frobenius number $g(a_1, \ldots, a_k)$ and the number of *omitted* values ω that cannot be decomposed over $a_1, \ldots, a_k$ [5]:

$$g := g(a_1, \ldots, a_k) = \max_p\{n_p\} - a_1 \quad \text{and} \quad \omega = \sum_p \left\lfloor \frac{n_p}{a_1} \right\rfloor = \frac{1}{a_1} \sum_p n_p - \frac{a_1 - 1}{2}.$$

Many algorithms for computing the Frobenius number rely on the above result. For example, Davison' algorithm [6] makes implicit use of this table. To explicitly compute the values n_p for $p = 0, \ldots, a_1 - 1$, Nijenhuis [14] gave an algorithm with runtime $O(k\, a_1 \log a_1)$, where the $\log a_1$ factor is due to a binary heap structure that must be updated in every step. One can easily modify Nijenhuis' algorithm by using a Fibonacci heap instead of a binary heap, thereby reaching a $O(a_1(k + \log a_1))$ runtime bound, but the constant factor overhead (runtime and memory) is much higher for a Fibonacci heap.

3 The Round Robin Algorithm

We compute the values n_p, for $p = 0, \ldots, a_1 - 1$, iteratively for the sub-problems "Find n_p for the instance $a_1, \ldots, a_i$" for $i = 1, \ldots, k$. For $i = 1$ we start with $n_0 = 0$ and $n_p = \infty$, $p = 1, \ldots, a_1 - 1$.

Suppose we know the correct values n'_p for the sub-problem $a_1, \ldots, a_{k-1}$ and want to calculate those of the original problem $a_1, \ldots, a_k$. We first concentrate on the simple case that $\gcd(a_1, a_k) = 1$. We initialize $n_p \leftarrow n'_p$ for all $p = 0, \ldots, a_1 - 1$ and $n \leftarrow n_0 = 0$. In every step of the algorithm, set $n \leftarrow n + a_k$ and $p \leftarrow n \bmod a_1$. Let $n \leftarrow \min\{n, n_p\}$ and $n_p \leftarrow n$. We repeat this loop until n equals 0.

p	$a_1 = 5$	$a_2 = 8$	$a_3 = 9$	$a_4 = 12$
0	0	0	0	0
1	∞	16	16	16
2	∞	32	17	12
3	∞	8	8	8
4	∞	24	9	9

Fig. 1. Table n_p for the MCP instance $5, 8, 9, 12$

In case all $a_2, \ldots, a_k$ are coprime to a_1, this short algorithm is already sufficient to find the correct values n_p. We have displayed a small example in Figure 1, where every column corresponds to one iteration of the algorithm as described in the previous paragraph. For example, focus on the column $a_3 = 9$. We start with $n = 0$. In the first step, we have $n \leftarrow 9$ and $p = 4$. Since $n < n_4 = 24$ we update $n_4 \leftarrow 9$. Second, we have $n \leftarrow 9 + 9 = 18$ and $p = 3$. In view of $n > n_3 = 8$ we set $n \leftarrow 8$. Third, we have $n \leftarrow 8 + 9 = 17$ and $p = 2$. Since $n < n_2 = 32$ we update $n_2 \leftarrow 17$. Fourth, we have $n \leftarrow 17 + 9 = 26$ and $p = 1$. In view of $n > n_1 = 16$ we set $n \leftarrow 16$. Finally, we return to $p = 0$ via $n \leftarrow 16 + 9 = 25$.

From the last column we can see that the Frobenius number for this example is $g(5, 8, 9, 12) = 16 - 5 = 11$.

It is straighforward how to generalize the algorithm for $d := \gcd(a_1, a_i) > 1$: In this case, we do the updating independently for every residue $r = 0, \ldots, d-1$: Only those n_p for $p \in \{0, \ldots, a_1 - 1\}$ are updated that satisfy $p \equiv r \pmod{d}$. To guarantee that the round robin loop completes updating after a_1/d steps, we have to start the loop from a minimal n_p with $p \equiv r \pmod{d}$. For $r = 0$ we know that $n_0 = 0$ is the unique minimum, while for $r \neq 0$ we search for the minimum first.

Round Robin Algorithm

```
 1  initialize n_0 = 0 and n_p = ∞ for p = 1, ..., a_1 − 1
 2  for i = 2, ..., k do
 3      d = gcd(a_1, a_i);
 4      for r = 0, ..., d − 1 do
 5          Find n = min{n_q | q mod d = r, 0 ≤ q ≤ a_1 − 1}
 6          If n < ∞ then repeat a_1/d − 1 times
 7              n ← n + a_i; p = n mod a_1;
 8              n ← min{n, n_p}; n_p ← n;
 9          done;
10      done;
11  done.
```

The inner loop (lines 6–9) will be executed only if the minimum $\min\{n_q\}$ is finite; otherwise, the elements of the residue class cannot be decomposed over $a_1, \ldots, a_i$ because of $\gcd(a_1, \ldots, a_i) > 1$.

Lemma 1. *Suppose that n'_p for $p = 0, \ldots, a_1 - 1$ are the correct residue table entries for the MCP instance $a_1, \ldots, a_{k-1}$. Initialize $n_p \leftarrow n'_p$ for $p = 0, \ldots, a_1 - 1$.*

Then, after one iteration of the outer loop (lines 3–10) of the Round Robin Algorithm, the residue table entries equal the values n_p for $p = 0, \ldots, a_1 - 1$ for the MCP instance $a_1, \ldots, a_k$.

Since for $k = 1$, $n_0 = 0$ and $n_p = \infty$ for $p \neq 0$ are the correct values for the MCP with one coin, we can use induction to show the correctness of the algorithm. To prove the lemma, we first note that for all $p = 0, \ldots, a_1 - 1$,

$$n_p \leq n'_p \quad \text{and} \quad n_p \leq n_q + a_k \text{ for } q = (p - a_k) \bmod a_1$$

after termination. Assume that for some n, there exists a decomposition $n = \sum_{i=1}^{k} a_i x_i$. We have to show $n \geq n_p$ for $p = n \bmod a_1$. Now, $\sum_{i=1}^{k-1} a_i x_i = n - a_k x_k$ is a decomposition of the MCP instance $a_1, \ldots, a_{k-1}$ and for $q = (n - a_k x_k) \bmod a_1$ we have $n - a_k x_k \geq n'_q$. We conclude

$$n_p \leq n_q + a_k x_k \leq n'_q + a_k x_k \leq n.$$

By an analogous argument, we infer $n_p = n$ for *minimal* such n. One can easily show that $n_p = \infty$ if and only if no n with $n \equiv p \pmod{a_1}$ has a decomposition with respect to the MCP instance $a_1, \ldots, a_k$.

Under the standard model of computation, time and space complexity of the algorithm are immediate and we reach:

Theorem 1. *The Round Robin Algorithm computes the residue table of an instance $a_1, \ldots, a_k$ of the Money Changing Problem, in runtime $\Theta(k\, a_1)$ and extra memory $O(1)$.*

To obtain a decomposition of any n in k steps, we save for every $p = 0, \ldots, a_1 - 1$ the minimal index i such that a decomposition $x_1, \ldots, x_k$ of n_p has $x_i > 0$, and we also save the maximal x_i for any such decomposition. This can be easily incorporated into the algorithm retaining identical time complexity, and requires $2a_1$ additional words of memory. Doing so, we obtain a lexicographically maximal decomposition. Note that unlike the Change Making Problem, we do not try to minimize the number of coins used.

4 Heuristic Runtime Improvements

We can improve the Round Robin Algorithm in the following ways: First, we do not have to explicitly compute the greatest common divisor $\gcd(a_1, a_i)$. Instead, we do the first round robin loop (lines 6–9) for $r = 0$ with $n = n_0 = p = 0$ until we reach $n = p = 0$ again. We count the number of steps t to this point. Then, $d = \gcd(a_1, a_i) = \frac{a_1}{t}$ and for $d > 1$, we do the remaining round robin loops $r = 1, \ldots, d - 1$.

Second, for $r > 0$ we do not have to explicitly search for the minimum in $N_r := \{n_q : q = r, r + d, r + 2d, \ldots, r + (a_1 - d)\}$. Instead, we start with $n = n_r$ and do exactly $t - 1$ steps of the round robin loop. Here, $n_r = \infty$ may hold, so

we initialize $p = r$ (line 5) and update $p \leftarrow (p + a_i) \bmod a_1$ separately in line 7. Afterwards, we continue with this loop until we first encounter some $n_p \leq n$ in line 8, and stop there. The second loop takes at most $t-1$ steps, because at some stage we reach the minimal $n_p = \min N_r$ and then, $n_p < n$ must hold because of the minimality of n_p. This compares to the t steps for finding the minimum.

Third, A. Nijenhuis suggested the following improvement (personal communication): Suppose that k is large compared to a_1, for example $k = O(a_1)$. Then, many round robin loops are superfluous because chances are high that some a_i is representable using $a_1, \ldots, a_{i-1}$. To exclude such superfluous loops, we can check after line 2 whether $n_p \leq a_i$ holds for $p = a_i \bmod a_1$. If so, we can skip this a_i and continue with the next index $i+1$, since this implies that a_i has a decomposition over $a_1, \ldots, a_{i-1}$. In addition, this allows us to find a minimal subset of $\{a_1, \ldots, a_k\}$ sufficient to decompose any number that can be decomposed over the original MCP instance $a_1, \ldots, a_k$.

Fourth, if $k \geq 3$ then we can skip the round robin loop for $i = 2$: The Extended Euclid's Algorithm [12] computes integers d, u_1, u_2 such that $a_1 u_1 + a_2 u_2 = d = \gcd(a_1, a_2)$. Hence, for the MCP instance a_1, a_2 we have $n_p = \frac{1}{d}((p\, a_2 u_2) \bmod (a_1 a_2))$ for all $p \equiv 0 \pmod{d}$, and $n_p = \infty$ otherwise. Thus, we can start with the round robin loop for $i = 3$ and compute the values n'_p of the previous instance a_1, a_2 on the fly using the above formula.

5 Cache Optimizations

Our last improvement is based on the following observation: The residue table $(n_p)_{p=0,\ldots,a_1-1}$ is very memory consuming, and every value n_p is read (and eventually written) exactly once during any round robin loop. Nowadays processors usually have a layered memory access model, where data is temporarily stored in a cache that has much faster access times than the usual memory. The following modification of the Round Robin Algorithm allows for cache-optimized access of the residue table: We exchange the $r = 0, \ldots, \gcd(a_1, a_i) - 1$ loop (lines 4–10) with the inner round robin loop (lines 6–9). In addition, we make use of the second improvement introduced above and stop the second loop as soon as *none* of the n_{p+r} was updated. Now, we may assume that the consecutive memory access of the loop over r runs in cache memory.

We want to roughly estimate how much runtime this improvement saves us. For two random numbers u, v drawn uniformly from $\{1, \ldots, n\}$, the expected value of the greatest common divisor is approximately $\mathbb{E}\big(\gcd(u,v)\big) \approx 6/\pi^2 H_n$, where H_n is the n'th harmonic number [12]. This leads us to the approximation $\mathbb{E}\big(\gcd(u,v)\big) \approx 1.39 \cdot \log_{10} n + 0.35$, so the expected greatest common divisor grows logarithmically with the coin values, for random input[2]. Even so, the improvement has relatively small impact on average: Let t_{mem} denote the runtime of our algorithm in main memory, and t_{cache} the runtime for the same

[2] For simplicity, we ignore the fact that due to the sorting of the input, $a_1 = \min\{a_1, \ldots, a_k\}$ is *not* drawn uniformly, and we also ignore the dependence between the drawings

instance of the problem if run in cache memory. For an instance $a_1, \ldots, a_k$ of MCP, the runtime t_{mod} of our modified algorithm roughly depends on the values $1/\gcd(a_1, a_i)$:

$$t_{\text{mod}} \approx t_{\text{cache}} + (t_{\text{mem}} - t_{\text{cache}}) \frac{1}{k-1} \sum_{i=2}^{k} \frac{1}{\gcd(a_1, a_i)}.$$

For random integers u, v uniformly drawn from $\{1, \ldots, n\}$ we estimate (analogously to [12], Section 4.5.2)

$$\mathbb{E}\left(\frac{1}{\gcd(u,v)}\right) \approx \sum_{d=1}^{n} \frac{p}{d^2}\frac{1}{d} = p \sum_{d=1}^{n} \frac{1}{d^3} = pH_n^{(3)} \quad \text{with } p = 6/\pi^2,$$

where the $H_n^{(3)}$ are the harmonic numbers of third order. The $H_n^{(3)}$ form a monotonically increasing series with $1.2020 < H_n^{(3)} < H_\infty^{(3)} < 1.2021$ for $n \geq 100$, so $\mathbb{E}\big(1/\gcd(u,v)\big) \approx 0.731$. If we assume that accessing main memory is the main contributor to the overall runtime of the algorithm, then we reduce the overall runtime by roughly one fourth. This agrees with our runtime measurements for random input as reported in the next section.

Round Robin Algorithm (optimized version for $k \geq 3$)

```
1  Initialize n_0, ..., n_{a_1 - 1} for the instance a_1, a_2, a_3; // fourth improvement
2  For i = 4, ..., k do
3      If n_p ≤ a_i for p = a_i mod a_1 then continue with next i; // third improvement
4      d = gcd(a_1, a_i);
5      p ← 0; q ← a_i mod a_1; // p is source residue, q is destination residue
6      Repeat a_1/d − 1 times
7          For r = 0, ..., d − 1 do
8              n_{q+r} ← min{n_{q+r}, n_{p+r} + a_i};
9          done;
10         p ← q; q ← (q + a_i) mod a_1;
11     done;
12     // update remaining entries, second improvement
13     Repeat
14         For r = 0, ..., d − 1 do
15             n_{q+r} ← min{n_{q+r}, n_{p+r} + a_i};
16         done;
17         p ← q; q ← (q + a_i) mod a_1;
18     until no entry n_{q+r} was updated;
19 done.
```

Fig. 2. Optimized version of the Round Robin Algorithm

In Figure 2 we have incorporated all but the first improvements into the Round Robin Algorithm; note that the first and last improvement cannot be incorporated simultaneously. All presented improvements are runtime heuristics, so the resulting algorithm still has runtime $O(k\, a_1)$.

6 Computational Results

We generated 12 000 random instances of MCP, with $k = 5, 10, 20$ and $10^3 \leq a_i \leq 10^7$. We have plotted the runtime of the optimized Round Robin Algorithm against a_1 in Figure 3. As expected, the runtime of the algorithm is mostly independent of the structure of the underlying instance. The processor cache, on the contrary, is responsible for major runtime differences. The left plot contains only those instances with $a_1 \leq 10^6$; here, the residue table of size a_1 appears to fit into the processor cache. The right plot contains all random instances; for $a_1 > 10^6$, the residue table has to be stored in main memory.

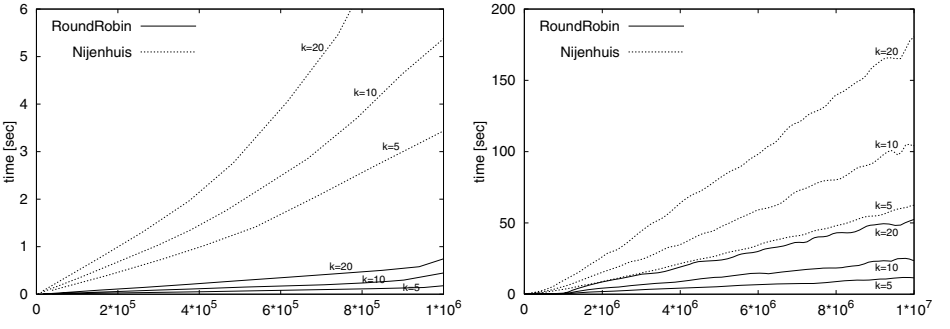


Fig. 3. Runtime vs. a_1 for $k = 5, 10, 20$ where $a_1 \leq 10^6$ (left) and $a_1 \leq 10^7$ (right)

To compare runtimes of the Round Robin Algorithm with those of Nijenhuis' algorithm [14], we re-implemented the latter in C++ using a binary heap as the priority queue. As one can see in Figure 3, the speedup of our optimized algorithm is about 10-fold for $a_1 \leq 10^6$, and more than threefold otherwise. Regarding Kannan's algorithm [11], the runtime factor $k^{k^k} > 10^{2184}$ for $k = 5$ makes it impossible to use this approach.

Comparing the original Round Robin Algorithm with the optimized version, the achieved speedup was 1.67-fold on average (data not shown).

We also tested our algorithm on some instances of the Money Changing Problem that are known to be "hard": Twenty-five examples with $5 \leq k \leq 10$ and $3719 \leq a_1 \leq 48709$ were taken from [2], along with runtimes given there for standard linear programming based branch-and-bound search. The runtime of the optimized Round Robin Algorithm (900 MHz UltraSparc III processor, C++) for every instance is below 10 ms, see Table 1. Note that in [2], Aardal and Lenstra do not compute the Frobenius number g but only verify that g cannot be decomposed. In contrast, the Round Robin Algorithm computes the residue table of the instance, which in turn allows to answer *all* subsequent questions whether some n is decomposable, in constant time. Still and all, runtimes usually compare well to those of [2] and clearly outperform LP-based branch-and-bound search (all runtimes above 9000 ms), taking into account the threefold processor power running the Round Robin Algorithm.

Table 1. Runtimes on instances from [2] in milliseconds, measured on a 359 MHz UltraSparc II (Aardal & Lenstra) and on a 900 MHz UltraSparc III (Round Robin)

Instance	c1	c2	c3	c4	c5	p1	p2	p3	p4	p5	p6	p7	p8
Aardal & Lenstra	1	1	1	1	1	1	1	2	1	2	1	2	1
Round Robin	0.8	0.9	0.9	1.1	1.6	3.1	1.2	4.9	9.2	4.0	3.4	3.1	2.8

Instance	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18	p19	p20
Aardal & Lenstra	3	2	5	12	6	12	80	80	150	120	100	5
Round Robin	0.4	6.5	1.8	2.1	2.4	1.8	2.1	6.4	2.5	3.7	3.2	9.0

7 Conclusion

We have presented the Round Robin Algorithm that generates a residue table which allows us to find the Frobenius number, as well as to answer whether any number n can be decomposed, the latter in constant time. The advantages of our algorithm are (i) its simplicity, making it easy to implement and allowing further improvements; (ii) its guaranteed worst case runtime of $O(k\,a_1)$, independent of the structure of the underlying instance; and (iii) its constant extra memory requirements. To the best of our knowledge, no other algorithm with worst case runtime $O(k\,a_1)$ is known. In addition, runtimes of the Round Robin Algorithm compare well to other, more sophisticated approaches, see [3]. It is rather surprising that despite the simplicity and efficiency of the Round Robin algorithm, it has not been reported in literature.

Simulations clearly show that the time consuming part of the Round Robin algorithm is accessing memory. We are currently working on a modification that performs cache-optimized memory access as described in Section 5, even when $\gcd(a_1, a_i)$ is small.

As mentioned in the introduction, we can use a slightly modified version of the Round Robin Algorithm to compute a data structure, which in turns allows us to find *all* decompositions of any input n. To this end, we generate an *extended residue table* of size $O(ka_1)$ in runtime $O(ka_1)$ that stores not only the "final" column of the residue table, but also all intermediate steps, confer Fig. 1. If we denote the number of decompositions of n by $\gamma(n)$, then this data structure allows us to generate all decompositions in time $O(ka_1\gamma(n))$ backtracking through the extended residue table, see [4].

Acknowledgments

Implementation and simulations by Henner Sudek. We thank Stan Wagon and Albert Nijenhuis for helpful discussions.

References

1. K. Aardal, C. Hurkens, and A. K. Lenstra. Solving a system of diophantine equations with lower and upper bounds on the variables. *Math. Operations Research*, 25:427–442, 2000.

2. K. Aardal and A. K. Lenstra. Hard equality constrained integer knapsacks. *Lect. Notes Comput. Sc.*, 2337:350–366, 2002.
3. D. E. Beihoffer, J. Hendry, A. Nijenhuis, and S. Wagon. Faster algorithms for Frobenius numbers. In preparation.
4. S. Böcker and Zs. Lipták. Efficient mass decomposition. In *Proc. of ACM Symposium on Applied Computing 2005*, pages 151–157, Santa Fe, USA, 2005.
5. A. Brauer and J. E. Shockley. On a problem of Frobenius. *J. Reine Angew. Math.*, 211:215–220, 1962.
6. J. L. Davison. On the linear diophantine problem of Frobenius. *J. Number Theory*, 48(3):353–363, 1994.
7. E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
8. R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, Massachusetts, second edition, 1994.
9. H. Greenberg. Solution to a linear diophantine equation for nonnegative integers. *J. Algorithms*, 9(3):343–353, 1988.
10. B. R. Heap and M. S. Lynn. A graph-theoretic algorithm for the solution of a linear diophantine problem of Frobenius. *Numer. Math.*, 6:346–354, 1964.
11. R. Kannan. Lattice translates of a polytope and the Frobenius problem. *Combinatorica*, 12:161–177, 1991.
12. D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, Massachusetts, third edition, 1997.
13. G. S. Lueker. Two NP-complete problems in nonnegative integer programming. Technical Report TR-178, Department of Electrical Engineering, Princeton University, March 1975.
14. A. Nijenhuis. A minimal-path algorithm for the "money changing problem". *Amer. Math. Monthly*, 86:832–835, 1979. Correction in *Amer. Math. Monthly*, 87:377, 1980.
15. J. L. Ramírez-Alfonsín. Complexity of the Frobenius problem. *Combinatorica*, 16(1):143–147, 1996.
16. J. L. Ramírez-Alfonsín. *The Diophantine Frobenius Problem*. Oxford University Press, 2005. To appear.
17. J. J. Sylvester and W. J. Curran Sharp. Problem 7382. *Educational Times*, 37:26, 1884.
18. H. S. Wilf. A circle-of-lights algorithm for the "money-changing problem". *Amer. Math. Monthly*, 85:562–565, 1978.
19. J. W. Wright. The change-making problem. *J. Assoc. Comput. Mach.*, 22(1):125–128, 1975.

W-Hardness Under Linear FPT-Reductions: Structural Properties and Further Applications*

Jianer Chen[1], Xiuzhen Huang[2], Iyad A. Kanj[3], and Ge Xia[4]

[1] Dept. of Computer Science, Texas A&M University, College Station, TX 77843
College of Information Science and Engineering, Central South University
Changsha 410083, P.R. China
chen@cs.tamu.edu

[2] Computer Science Department, Arkansas State University, State University, Arkansas 72467, USA
xzhuang@csm.astate.edu

[3] School of Computer Science, Telecommunications and Information Systems
DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604-2301, USA
ikanj@cs.depaul.edu

[4] Department of Computer Science, Lafayette College, Easton, PA 18042, USA
gexia@cs.lafayette.edu

Abstract. The notion of linear fpt-reductions has been recently used to derive strong computational lower bounds for well-known NP-hard problems. In this paper, we formally investigate the notions of $W[t]$-hardness and $W[t]$-completeness under the linear fpt-reduction, and study structural properties of the corresponding complexity classes. Additional complexity lower bounds on important computational problems are also established.

1 Introduction

A *parameterized problem* Q is a decision problem consisting of instances of the form (x, k), where the integer $k \geq 0$ is called the *parameter*. The parameterized problem Q is *fixed-parameter tractable* [8] if it can be solved in time $f(k)|x|^{O(1)}$, where f is a recursive function[1]. Certain NP-hard parameterized problems, such as VERTEX COVER, are fixed-parameter tractable, and hence can be solved practically for small parameter values [7]. On the other hand, the inherent computational difficulty of solving many other NP-hard parameterized problems with even small parameter values has suggested that certain parameterized problems be not fixed-parameter tractable, which has motivated the *theory of fixed-parameter intractability* [8]. The W-hierarchy $\bigcup_{t\geq 0} W[t]$ has been introduced to characterize the inherent level of intractability for parameterized problems.

* This research is supported in part by US NSF under Grants CCR-0311590 and CCF-0430683, by China NNSF under Grants No.60373083 and No.60433020, and by DePaul University Competitive Research Grant

[1] In this paper, we always assume that complexity functions are "nice" with both domain and range being non-negative integers and the values of the functions and their inverses can be easily computed

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 975–984, 2005.

A large number of parameterized problems have been proved to be hard or complete for various levels in the W-hierarchy [8]. Examples of $W[1]$-hard problems include many well-known NP-hard problems such as CLIQUE, DOMINATING SET, SET COVER, and WEIGHTED CNF SATISFIABILITY. The theory of parameterized intractability has found important applications in a variety of areas such as database systems and model checking [9, 10, 16].

The $W[1]$-hardness of a parameterized problem provides a strong evidence that the problem is not solvable in time $f(k)n^{O(1)}$ for any function f. However, $W[1]$-hardness dose not provide further information on how precisely the problem complexity depends on the parameter k. For example, the $W[1]$-hardness of the CLIQUE problem does not exclude the possibility of solving the problem in time $O(n^{\log\log k})$. Note that such an algorithm would be practically acceptable for moderate values of the parameter k, such as $k = 1000$.

Recent investigation has started along this line of research. In particular, the concept of *linear fpt-reduction* has been introduced to derive stronger computational lower bounds for well-known NP-hard parameterized problems [4, 5]. For example, based on the linear fpt-reduction, it has been shown that unless an unlikely collapse occurs in the parameterized complexity theory, any algorithm solving the CLIQUE problem takes time at least $n^{\Omega(k)}$. Note that this lower bound is asymptotically tight in the sense that the trivial algorithm that enumerates all subsets of k vertices in a given graph to test the existence of a clique of size k runs in time $O(n^k)$.

Therefore, the linear fpt-reduction has provided a powerful method for deriving strong computational lower bounds. In this paper, we formally investigate the concepts of $W[t]$-hardness and $W[t]$-completeness under the linear fpt-reduction, and systematically study the structural properties of the corresponding complexity classes. These complexity classes are defined based on generic complete problems under the linear fpt-reduction, instead of on computational models as for most traditional complexity classes. Therefore, it is natural to ask whether the familiar structural properties for traditional complexity hierarchies still hold true for the new parameterized complexity hierarchy. Moreover, the study of the structural properties of the new complexity classes has a direct impact on the applications of the theory to derive strong computational lower bounds.

We then illustrate the power of our techniques by deriving complexity lower bounds for further computational problems. We note that many of the fpt-reductions proposed in the literature are actually linear fpt-reductions or can be easily modified to become linear fpt-reductions. This enables us to quickly expand the list of computational problems with strong complexity lower bounds. We also study the parameterized complexity of the problems in the classes LOGNP and LOGSNP introduced by Papadimitriou and Yannakakis [15]. These problems are solvable in time $n^{O(\log n)}$ and therefore look "easier" than NP-hard problems in general. In particular, we study the problems TOURNAMENT DOMINATING SET, RICH HYPERGRAPH COVER, and V-C DIMENSION in these classes, and prove that, these problems are $W[1]$-hard under the linear fpt-reduction. In consequence, unless an unlikely collapse occurs in the parameterized complexity theory, these problems cannot be solved in time $f(k)n^{o(k)}$, neither can the optimization ver-

sions of these problems have polynomial time approximation schemes of running time $f(1/\epsilon)n^{o(1/\epsilon)}$, for any function f. These results either improve or complement previous research on the problems, and advance our understanding on the complexity of the problems.

We briefly review the related terminologies. Denote by FPT the class of all fixed-parameter tractable problems. A circuit C is a *Π_t-circuit* if its output gate is an AND gate and it has depth t. The *weight* of an assignment τ to a circuit is the number of variables assigned value 1 by τ. The parameterized problem WEIGHTED SATISFIABILITY on Π_t-circuits, abbreviated WCS$[t]$, is to determine for a given Π_t-circuit C and an integer k, whether C has a satisfying assignment of weight k. The WEIGHTED MONOTONE SATISFIABILITY (resp. WEIGHTED ANTIMONOTONE SATISFIABILITY) problem on Π_t-circuits, abbreviated WCS$^+[t]$ (resp. WCS$^-[t]$) is defined similarly as WCS$[t]$ except that the circuit C is required to be monotone (resp. antimonotone). To simplify our statements, we will denote by WCS$^*[t]$ the problem WCS$^+[t]$ if t is even and the problem WCS$^-[t]$ if t is odd. Finally, the WEIGHTED ANTIMONOTONE CNF 2SAT problem, abbreviated WCNF 2SAT$^-$, consists of the pairs of the form (F, k), where k is an integer k and F is a CNF formula in which all literals are negative and each clause contains at most 2 literals, such that F has a satisfying assignment of weight k.

Due to the space limit, proofs for Theorems 1, 2, 3, and 5 have been omitted. We refer interested readers to the full version of the paper [6].

2 W_l-Hardness and W_l-Completeness

Each instance (C, k) of the WCS$[t]$ problem can be regarded as a search problem, in which we need to select k elements from a search space consisting of a set of n input variables, and assign them value 1 so that the circuit C is satisfied. Many well-known computational problems, such as WEIGHTED CNF SAT, SET COVER, and HITTING SET, have similar formulations. The interested reader is referred to [5] for detailed discussion on this issue.

We will concentrate on parameterized problems that seek a subset in a search space satisfying certain properties. Thus, each instance of our parameterized problem is associated with a triple (k, n, m), where k is the parameter, n is the size of the search space, and m is the instance size[2].

Definition 1. A parameterized problem Q is *linear fpt-reducible*, shortly *fpt$_l$-reducible*, to a parameterized problem Q' if there exist a function f and an algorithm A of running time $f(k)m^{O(1)}$ that, on each (k, n, m)-instance x of Q, produces a (k', n', m')-instance x' of Q', where $k' = O(k)$, $n' = n^{O(1)}$, $m' = m^{O(1)}$, and x is a yes-instance of Q if and only if x' is a yes-instance of Q'.

It is easy to verify that the fpt$_l$-reducibility is transitive. Similar to the W-hierarchy defined in terms of the standard fpt-reducibility [8], we introduce a W-hierarchy based on the fpt$_l$-reducibility.

[2] For most problems in our consideration, the search space can be easily identified. For problems in which the search space is not easily identified, we simply let $n = m$

Definition 2. A parameterized problem Q is $W[1]$*-hard under the fpt$_l$-reduction*, shortly $W_l[1]$*-hard*, if the WCNF 2SAT$^-$ problem is fpt$_l$-reducible to Q. The problem Q is $W[t]$*-hard under fpt$_l$-reduction*, shortly $W_l[t]$*-hard*, for $t \geq 2$ if the WCS$^*[t]$ problem is fpt$_l$-reducible to Q. For all $t \geq 1$, a parameterized problem Q is $W_l[t]$*-complete* if Q is in $W[t]$ and is $W_l[t]$-hard.

The $W_l[t]$-hardness has been used to derive strong complexity lower bounds. For $W_l[t]$-hard problems where $t \geq 2$, we have the following result.

Proposition 1. (Theorem 5.1, [5]) *For any integer $t \geq 2$, unless $W[t-1] = FPT$, no $W_l[t]$-hard problem can be solved in time $f(k)n^{o(k)}m^{O(1)}$ for any recursive function f.*

Computational lower bounds for $W_l[1]$-hard problems have been closely related to the *exponential time hypothesis* (ETH), which was first articulated in [12]. This hypothesis conjectures that the problem 3-SATISFIABILITY cannot be solved in time $2^{o(n)}$. To support the hypothesis, Impagliazzo and Paturi [12] have shown that if ETH fails then many well-known NP-hard problems, including all SNP problems formulated in [14], are solvable in subexponential time. Note that many of the SNP problems have been the targets for exact algorithms for decades but no subexponential time algorithms for them have been developed.

It is known [8] that ETH implies $W[1] \neq FPT$.

Proposition 2. (Theorem 5.2, [5]) *Unless ETH fails, no $W_l[1]$-hard problem is solvable in time $f(k)m^{o(k)}$ for any recursive function f.*

The main result in this section is that for any $t \geq 1$, $W_l[t+1]$-hardness implies $W_l[t]$-hardness. There are a number of reasons why this result is not trivial and should be examined carefully:

- In most hierarchies in complexity theory, the hardness for an upper level implies trivially the hardness for a lower level. For example, a Σ_{t+1}^p-hard problem in the polynomial time hierarchy is automatically Σ_t^p-hard by the definitions [13]. Therefore, it will be interesting to check whether such a common property is also shared by the W_l-hierarchy.
- Such a result does not trivially follow from the definitions. The $W_l[t]$-hardness is defined differently according to the parity of the integer t: for an even integer t, $W_l[t]$-hardness is defined based on the satisfiability problem WCS$^+[t]$ on monotone circuits, while for an odd integer t, $W_l[t]$-hardness is defined based on the satisfiability problem WCS$^-[t]$ on antimonotone circuits. In particular, the fpt-reduction from the problem WCS$^*[t-1]$ to the problem WCS$^*[t]$ proposed in the literature [8] is *not* a linear fpt-reduction (see Chapter 12 in [8] for details).
- Note that the lower bound in Proposition 2 is actually stronger than that in Proposition 1 since the search space size n is in general not larger than the instance size m. That is, $W_l[1]$-hardness in fact implies a stronger lower bound (although also under a stronger working hypothesis) than that implied by $W_l[t]$-hardness for $t > 1$. Therefore, proving that $W_l[t]$-hardness implies $W_l[t-1]$-hardness will immediately provide a stronger computational lower bound for $W_l[t]$-hard problems when $t > 1$.

Theorem 1. *For any $t \geq 2$, $W_l[t+1]$-hardness implies $W_l[t]$-hardness.*

Theorem 2. *$W_l[2]$-hardness implies $W_l[1]$-hardness.*

3 New Lower Bounds

Propositions 1 and 2 offer powerful techniques for deriving strong complexity lower bounds for well-known NP-hard problems. In particular, it has been shown [4, 5] that the following parameterized problems are $W_l[2]$-hard: WEIGHTED CNF SATISFIABILITY, SET COVER, HITTING SET, and DOMINATING SET, and that the following parameterized problems are $W_l[1]$-hard: WEIGHTED CNF q-SAT for any integer $q \geq 2$, CLIQUE, and INDEPENDENT SET. According to Proposition 2, none of these problems can be solved in time $f(k)m^{o(k)}$ for any recursive function f unless ETH fails.

In this section we expand the list of $W_l[1]$-hard problems by developing linear fpt-reductions from the known $W_l[1]$-hard problems. In fact, many existing fpt-reductions proposed in the literature are linear fpt-reductions. Therefore, these fpt-reductions can be directly used or modified for our purpose. Using this approach, we can quickly get a much longer list of $W_l[1]$-hard problems and claim strong complexity lower bounds for these problems. The reader is referred to [8] for precise definitions for these problems.

Theorem 3. (1) *The following parameterized problems are $W_l[2]$-hard:* RED-BLUE DOMINATING SET, DOMINATING CLIQUE, PRECEDENCE CONSTRAINED PROCESSOR SCHEDULING, FEATURE SET, *and* WEIGHTED BINARY INTEGER PROGRAMMING*; and* (2) *The problem* SET PACKING *is $W_l[1]$-hard.*

In particular, none of these problems can be solved in time $f(k)m^{o(k)}$ for any recursive function f unless ETH fails.

Again, Theorem 3 gives asymptotically tight complexity lower bounds in a very strong sense for these well-known NP-hard problems. For example, even though the DOMINATING CLIQUE problem can be trivially solved by exhaustive enumeration in time $O(n^k m)$ of all subsets of k vertices, where n is the number of vertices and m is the instance size of the graph, solving the problem in time $f(k)m^{o(k)}$ is very unlikely for any recursive function f.

We further apply our technique to study two important problems in computational biology.

LONGEST COMMON SUBSEQUENCE: given a set $S = \{s_1, s_2, ..., s_k\}$ of k strings over a finite alphabet Σ, and an integer $\lambda > 0$, is there a string $s \in \Sigma^*$ of length λ, which is a subsequence of all of the k strings in S? Here the parameter is k.

SHORTEST COMMON SUPERSEQUENCE: given a set $S = \{s_1, s_2, ..., s_k\}$ of k strings over a finite alphabet Σ, and an integer $\lambda > 0$, is there a string $s \in \Sigma^*$ of length λ, which is a supersequence of all of the k strings in S? Here the parameter is k.

Theorem 4. *The problems* LONGEST COMMON SUBSEQUENCE *and* SHORTEST COMMON SUPERSEQUENCE *are* $W_l[1]$*-hard. In consequence, they cannot be solved in time* $f(k)m^{o(k)}$ *for any function* f*, unless ETH fails.*

Proof. Pietrzak [17] has recently proved the $W[1]$-hardness for the problems LONGEST COMMON SUBSEQUENCE and SHORTEST COMMON SUPERSEQUENCE by fpt-reductions from CLIQUE. For LONGEST COMMON SUBSEQUENCE, Pietrzak developed a polynomial time algorithm A_1 that, on an instance (G, k) of CLIQUE, produces an instance (S_1, λ_1, k_1) for LONGEST COMMON SUBSEQUENCE, where $k_1 = k + 1$ and $|S_1| = O(k^8 n^7) = O(n^{15})$, such that (G, k) is a yes-instance of CLIQUE if and only if (S_1, λ_1, k_1) is a yes-instance of LONGEST COMMON SUBSEQUENCE. This fpt-reduction is obviously a linear fpt-reduction. In consequence, the LONGEST COMMON SUBSEQUENCE problem is $W_l[1]$-hard. Note that here we have simply let the search space size in an instance of LONGEST COMMON SUBSEQUENCE to be equal to the instance size.

The other polynomial time algorithm developed by Pietrzak transforms an instance (G, k) of CLIQUE to an instance (S_2, λ_2, k_2) for SHORTEST COMMON SUPERSEQUENCE, where $k_2 = k + 1$ and $|S_2| = O(k^8 n^7) = O(n^{15})$, which gives a linear fpt-reduction from CLIQUE to SHORTEST COMMON SUPERSEQUENCE. In consequence, SHORTEST COMMON SUPERSEQUENCE is $W_l[1]$-hard. □

To see the significance of Theorem 4, we quote a sentence from [17]:

> *Unless an unlikely collapse in the parameterized hierarchy occurs, this*[3] *rules out the existence of exact algorithms with running time* $f(k)n^{O(1)}$ *(i.e., exponential only in* k*) for those problems. This does not mean that there are no algorithms with much better asymptotic time-complexity than the known* $O(n^k)$ *algorithms based on dynamic programming, e.g., algorithms with running time* $n^{\sqrt{k}}$ *are not deemed impossible by our results.*

Therefore, Theorem 4 has strengthened the results in [17] significantly and advanced our understanding on the complexity of the problems: it is actually unlikely that the problems can be solved in time $n^{\gamma(k)}$ for *any* sublinear function $\gamma(k)$, and the known dynamic programming algorithms of running time $O(n^k)$ for the problems are actually asymptotically optimal.

4 On the Complexity of LOGNP and LOGSNP Problems

To further illustrate the power of our methods, we consider another group of computational problems introduced by Papadimitriou and Yannakakis [15].

A directed graph G is a *tournament* if between each pair of vertices in G, there is exactly one directed edge. A hypergraph H is a *rich hypergraph* if every edge in H is incident on at least half of the vertices in H. In their study for the

[3] This refers to the results proved in [17] that the problems LONGEST COMMON SUBSEQUENCE and SHORTEST COMMON SUPERSEQUENCE are $W[1]$-hard

complexity classes LOGNP and LOGSNP, Papadimitriou and Yannakakis [15] have in particular considered the following problems:

RICH HYPERGRAPH COVER: given a rich hypergraph $H = (V, E)$ and a parameter k, is there a subset C of k vertices in H such that every edge in H is incident on at least one vertex in C?

TOURNAMENT DOMINATING SET: given a tournament graph T, and a parameter k, is there a subset D of k vertices in T such that for each vertex v not in D, there is at least one vertex w in D and $[w, v]$ is a directed edge in T?

V-C DIMENSION: given a family $\mathcal{F}$ of subsets of a universe U, and a parameter k, is there a subset S of U such that $|S| = k$ and for each subset T of S, there is a set $C_T \in \mathcal{F}$ satisfying $S \cap C_T = T$?

It can be shown [15] that if the parameter value k is larger than $\log m$, where m is the instance size, then the answer to RICH HYPERGRAPH COVER and TOURNAMENT DOMINATING SET is always positive while the answer to V-C DIMENSION is always negative. Therefore, the problem instances of these problems become non-trivial only when $k \leq \log m$. In consequence, all these problems can be solved in time $O(m^{\log m})$. Hence, these problems are unlikely to be NP-hard. On the other hand, it is unknown whether any of these problems is solvable in polynomial time.

Theorem 5. *The problems* RICH HYPERGRAPH COVER *and* TOURNAMENT DOMINATING SET *are* $W_l[2]$*-hard, and the problem* V-C DIMENSION *is* $W_l[1]$*-hard. In consequence, they cannot be solved in time* $f(k)m^{o(k)}$ *for any function* f*, unless ETH fails.*

The approximability of the problems in Theorem 5 has drawn research interests recently [1, 2]. Recall that an NP optimization problem Q has a *polynomial time approximation scheme* if there is an approximation algorithm A_Q that takes a pair (x, ϵ) as input, where x is an instance of Q and $\epsilon > 0$ is a real number, and returns a solution y for x such that the approximation ratio of the solution y is bounded by $1 + \epsilon$, and for a fixed $\epsilon > 0$, the running time of the algorithm A_Q is bounded by a polynomial of $|x|$. The algorithm A_Q is a *fully polynomial time approximation scheme* for Q if the running time of A_Q is bounded by a polynomial of $1/\epsilon$ and $|x|$, and is an *efficient polynomial time approximation scheme* for Q [3] if the running time of Q is bounded by $f(1/\epsilon)|x|^{O(1)}$ for a function f.

An NP optimization problem Q can be systematically parameterized [5] into a parameterized problem $Para(Q)$, whose instances take the form (x, k) asking whether the optimal value for x is not larger than k (resp. not smaller than k) in case Q is a minimization (resp. maximization) problem.

Theorem 6. (Theorem 6.1, [5]) *If the parameterized version* $Para(Q)$ *of an NP optimization problem* Q *is* $W_l[1]$*-hard, then* Q *has no polynomial time approximation scheme of running time* $f(1/\epsilon)m^{o(1/\epsilon)}$ *for any recursive function* f*, unless ETH fails.*

Consider the following optimization problems.

RICH HYPERGRAPH COVER-OPT: given a rich hypergraph $H = (V, E)$, find a minimum set C of vertices such that each edge in H is incident on at least one vertex in C.

TOURNAMENT DOMINATING SET-OPT: given a tournament graph T, find a minimum set D of vertices such that for each vertex v not in D, there is at least one vertex $w \in D$ and $[w, v]$ is a directed edge in T.

V-C DIMENSION-OPT: given a family $\mathcal{F}$ of subsets of a universe U, find a maximum subset S of U such that for each subset T of S, there is a set $C_T \in \mathcal{F}$ satisfying $S \cap C_T = T$.

Theorem 7. *Unless ETH fails, none of the problems* RICH HYPERGRAPH COVER-OPT*,* TOURNAMENT DOMINATING SET-OPT*, and* V-C DIMENSION-OPT *has polynomial time approximation schemes of running time* $f(1/\epsilon)m^{o(1/\epsilon)}$ *for any recursive function* f*.*

Proof. The parameterized versions of these problems are just the corresponding parameterized problems in Theorem 5. The theorem follows immediately from Theorem 5 and Theorem 6. □

Theorem 7 improves or complements a number of previous results. Papadimitriou and Yannakakis [15] introduced the classes LOGNP and LOGSNP, and proved that RICH HYPERGRAPH COVER and TOURNAMENT DOMINATING SET are complete under the polynomial time reduction for the class LOGSNP, and that V-C DIMENSION is complete under the polynomial time reduction for the class LOGNP. These results hint that it is unlikely that these problems can be solved in polynomial time. Theorem 7 shows that these problems are not only difficult for being solved precisely in polynomial time, but also difficult for being solved approximately in polynomial time. Cai and Chen [1] showed that the parameterized version of every NP optimization problem with fully polynomial time approximation schemes is fixed-parameter tractable, and Cesati and Trevisan [3] extended this result and proved that the parameterized version of every NP optimization problem with efficient polynomial time approximation schemes is fixed-parameter tractable. As a consequence, these results plus the $W[1]$-hardness of the problems RICH HYPERGRAPH COVER, TOURNAMENT DOMINATING SET, and V-C DIMENSION imply that these problems have no fully or efficient polynomial time approximation schemes. Theorem 7 further strengthens these results by showing the impossibility for these problems to have polynomial time approximation schemes of running time $f(1/\epsilon)m^{o(1/\epsilon)}$ for any recursive function f. Cai, Juedes, and Kanj [2] studied the approximability of these problems and proved that TOURNAMENT DOMINATING SET and RICH HYPERGRAPH COVER cannot be approximated to a ratio $c > 1$ unless DOMINATING SET can be approximated to a ratio $2c$ in time $O(2^{n^\delta})$ for some $\delta < 1$. Theorem 7 complements this result by showing the inpracticability of approximation algorithms with small approximation ratio for TOURNAMENT DOMINATING SET and RICH HYPERGRAPH COVER

(under a different working hypothesis). Moreover, it was posed as an open problem in [2] to study the inapproximability of V-C DIMENSION, and Theorem 7 provides an answer to this question.

We point out that the results in Theorems 5 and 7 can be extended to many other problems, such as the problems LOG CLIQUE, LOG DOMINATING SET, LOG HYPERGRAPH COVER, LOG ADJUSTMENT, LOG CHORDLESS PATH studied in [15]. For detailed discussions, interested readers are referred to [11].

5 A Remark on the W-Hierarchy

In this section, we provide an interesting observation on the original W-hierarchy. Most complexity hierarchies have the "hierarchical collapsing property" so that the collapsing of a lower level in the hierarchy implies the collapsing of all higher levels. For example, if for any integer $t > 0$, the t-th level of the polynomial time hierarchy collapses to the $(t-1)$-st level, $\Sigma_t^p = \Sigma_{t-1}^p$, then the entire polynomial time hierarchy collapses to the $(t-1)$-st level: $\Sigma_{t+h}^p = \Sigma_{t-1}^p$ for all $h \geq 0$ [13]. Most important complexity hierarchies, such as the NC hierarchy, the AC hierarchy, and the Boolean hierarchy, share a similar collapsing result [13].

It has been a well-known open problem in parameterized complexity theory whether the W-hierarchy satisfies a similar collapsing result. In particular, we are interested in knowing whether the following result holds true for the W-hierarchy:

Collapsing. If $W[t] = FPT$ for an integer $t \geq 1$, then $W[h] = FPT$ for all integer $h \geq t$.

One would expect naturally that the collapsing results such as **Collapsing** hold true. In the following, we discuss the consequence of **Collapsing**.

Theorem 8. *If* **Collapsing** *holds true, then the problem* WCS$^*[t]$ *either cannot be solved in time* $f_1(k)n^{o(k)}m^{O(1)}$ *for any function* f_1*, or can be solved in time* $f_2(k)m^{O(1)}$ *for a fixed function* f_2*.*

Proof. Suppose that the problem WCS$^*[t]$ can be solved in time $f_1(k)n^{o(k)}m^{O(1)}$ for a function f_1. By Proposition 1, this implies that $W[t-1] = FPT$. By **collapsing**, this would imply $W[t] = FPT$. Since WCS$^*[t]$ is in $W[t]$, we derive that WCS$^*[t]$ can be solved in time $f_2(k)m^{O(1)}$ for a function f_2. □

Obviously, the problem WCS$^*[t]$ in Theorem 8 can be replaced by any $W_l[t]$-complete problem.

Note that if **Collapsing** can be proved, then the conclusion in Theorem 8 holds true *unconditionally*, not depending on any complexity assumptions such as P $\neq$ NP or $W[1] \neq FPT$. This would exclude the possibility that, for example, the complexity of the CLIQUE problem is in the order of $\Theta(n^{\sqrt{k}})$.

References

1. L. Cai and J. Chen, On fixed parameter tractability and approximability of NP optimization problems, *Journal of Computer and System Sciences 54*, pp. 465-474, (1997).
2. L. Cai, D. Juedes, and I. A. Kanj, Inapproximability of non NP-hard optimization problems, *Theoretical Computer Science 289*, pp. 553-571, (2002).
3. M. Cesati and L. Trevisan, On the efficiency of polynomial time approximation schemes, *Information Processing Letters 64*, pp. 165-171, (1997).
4. J. Chen, B. Chor, M. Fellows, X. Huang, D. Juedes, I. Kanj, and G. Xia, Tight lower bounds for certain parameterized NP-hard problems, *Proc. 19th Annual IEEE Conference on Computational Complexity* (CCC 2004), pp. 150-160, (2004). Journal version is to appear in *Information and Computation*.
5. J. Chen, X. Huang, I. Kanj, and G. Xia, Linear FPT reductions and computational lower bounds, *Proc. 36th Annual ACM Symposium on Theory of Computing* (STOC 2004), pp. 212-221, (2004).
6. J. Chen, X. Huang, I. Kanj, and G. Xia, W-hardness under linear FPT-reductions: structural properties and further applications, *Tech. Report*, Dept. Computer Science, Texas A&M University, (2005).
7. J. Chen, I. A. Kanj, and W. Jia, Vertex cover: further observations and further improvements, *Journal of Algorithms 41*, pp. 280-301, (2001).
8. R.G. Downey and M.R. Fellows, *Parameterized Complexity*, Springer-Verlag, 1999.
9. J. Flum and M. Grohe, Model-checking problems as a basis for parameterized intractability, *Proc. 19th IEEE Symposium on Logic in Computer Science*, (LICS'04), pp. 388-397, (2004).
10. M. Grohe, The parameterized complexity of database queries, *Proc. 20th ACM Symposium on Principles of Database Systems*, (PODS'01), pp. 82-92, (2001).
11. X. Huang, *Parameterized Complexity and Polynomial-time Approximation Schemes*, Ph.D. Dissertation, Department of Computer Science, Texas A&M University, December, 2004.
12. R. Impagliazzo and R. Paturi, Which problems have strongly exponential complexity? *Journal of Computer and System Sciences 63*, pp. 512-530, (2001).
13. C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley Pub., Reading, Mass., 1995.
14. C. H. Papadimitriou and M. Yannakakis, Optimization, approximation, and complexity classes, *Journal of Computer and System Sciences 43*, pp. 425-440, (1991).
15. C.H. Papadimitriou and M. Yannakakis, On limited nondeterminism and the complexity of VC dimension, *Journal of Computer and System Sciences 53*, pp. 161-170, (1996).
16. C.H. Papadimitriou and M. Yannakakis, On the complexity of database queries, *Journal of Computer and System Sciences 58*, pp. 407-427, (1999).
17. K. Pietrzak, On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems, *Journal of Computer and System Sciences 67*, pp. 757-771, (2003).

Some New Results on Inverse Sorting Problems*

Xiao Guang Yang[1] and Jian Zhong Zhang[2]

[1] Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, 100080, China
[2] Department of Mathematics, City University of Hong Kong, Hong Kong

Abstract. In this paper, we consider two types of inverse sorting problems. The first type is an inverse sorting problem under weighted Hamming distance with bound constraints, which can be solved in $O(n^2)$ time. The second type is a family of partial inverse sorting problems which we can further divide into two kinds, with free positions and with fixed positions. We show that both types of partial inverse sorting problems can be solved by a combination of a sorting problem and an inverse sorting problem.

Keywords: sorting problem, inverse problem, partial inverse problem, Hamming distance, optimization.

1 Introduction

Given a set of numbers, say $\{a_1, a_2, \cdots, a_n\}$, the sorting problem is to arrange these numbers in a non-decreasing order (or non-increasing order). Mathematically it is to find a permutation ω of $I = \{1, 2, \cdots, n\}$ such that

$$a_{\omega(1)} \leq a_{\omega(2)} \leq \cdots \leq a_{\omega(n)}.$$

Sorting is a basic problem in daily life. It has many applications in various combinatorial and management problems. It is also a building block for many heuristic algorithms. It is well-known that the sorting problem can be solved in $O(n \ln(n))$ time.

Conversely, an inverse sorting problem is to modify the numbers such that a given order of the modified numbers can form a non-decreasing (or non-increasing) sequence. Without loss of generality, an inverse sorting problem can be stated as follows:

$$\min C(x - a) \tag{1}$$

$$\text{s.t. } x_1 \leq x_2 \leq \cdots \leq x_n \tag{2}$$

where $C(x - a)$ is a (weighted) deviation of vector x from vector a, or the cost of changing a into x. Applications of the inverse sorting problems and some

* Supported by the Hong Kong Universities Grant Council (CERG CITYU 1153/01P, internal code number 9040883) and the National Key Research and Development Program of China (2002CB312004) and the National Natural Science Foundation of China (700221001, 70425004)

L. Wang (Ed.): COCOON 2005, LNCS 3595, pp. 985–992, 2005.

algorithms to handle inverse sorting problems under different cost functions can be found in [1, 2].

In this paper, we consider two types of inverse sorting problems. The first one is an inverse sorting problem under weighted Hamming distance. Note that so far in all studied inverse sorting problems, $C(x-a)$ is a continuous function about the changes $x-a$ of the given numbers. But sometimes for each number a_i, we might care about only whether it is changed, and without considering the magnitude of its change, i.e. what we are concerned about is to change as few numbers as possible. To be a bit more general, suppose that different numbers in the set $\{a_1, a_2, \cdots, a_n\}$ have different weights. Then $C(x-a)$ under such a circumstance can be written as $C(x-a) = \sum_{i=1}^{n} w_i h(x_i - a_i)$, where w_i is the weight associated with number a_i, $h(t)$ is a Hamming function which is defined by $h(0) = 0$ and $h(t) = 1$ for any $t \neq 0$. Following the terminology of [3, 4], we call the inverse sorting problem with such $C(x-a)$ *an inverse sorting problem under weighted Hamming distance*. In the next section, we will consider such an inverse problem with bound constraints, i.e. $|x_i - a_i| \leq b_i$ for each number a_i and present an $O(n^2)$ algorithm to solve the inverse problem.

The second type of inverse sorting problems that we consider in this paper is some partial inverse sorting problems. Here "partial inverse sorting" means that, given a fixed order among a part of the numbers $a_1, \cdots, a_n$, we want to modify the n numbers such that the modified numbers have a complete order which agrees with the given partial order. Mathematically, the partial inverse sorting (PIS) problem can be described as the following:

(PIS) Given a set of numbers $a_1, \cdots, a_n$ and a permutation π of a subset $J \subset I = \{1, 2, \cdots, n\}$, modify a to a^* such that there exists a permutation ω of the whole set I with $a^*_{\omega(1)} \leq a^*_{\omega(2)} \leq \cdots \leq a^*_{\omega(n)}$, and a^* and ω satisfy that

(1) the order of ω agrees with that of π over J.

(2) $C(a^* - a)$ is minimum.

To clarify this, in the above statement, $\omega(i) = k$ means that under the permutation ω, the number k in I is arranged to the $i-$th position. $\pi(i) = k$ has a similar meaning, but with respect to the numbers in set J. So, the condition (1) requests that if two numbers p, q in J have $p = \pi(i) = \omega(i')$ and $q = \pi(j) = \omega(j')$, then $(i-j)(i'-j') \geq 0$.

The partial inverse sorting problems are further classified into two sub-types.

(PIS-1) The partial inverse sorting problem with free positions, i.e. the positions of π within ω can be arbitrary.

(PIS-2) The partial inverse sorting problem with fixed positions, i.e. the positions of π within ω are fixed.

2 Inverse Sorting Problem Under Weighted Hamming Distance

In this section, we consider the inverse sorting problem under weighted Hamming distance with bound constraints for changes.

The corresponding inverse sorting problem can be stated as follows:

$$\min \sum_{i=1}^{n} w_i h(x_i - a_i) \tag{3}$$

$$\text{s.t. } x_1 \le x_2 \le \cdots \le x_n, \tag{4}$$

$$|x_i - a_i| \le b_i, i = 1, 2, \cdots, n. \tag{5}$$

To tackle this inverse problem, we introduce a concept first.

For the sequence $\{a_1, a_2, \cdots, a_n\}$, a subsequence $\{a_{i_1}, a_{i_2}, \cdots, a_{i_k}\}$ is called a well-ordered subsequence if $a_{i_1} \le a_{i_2} \le \cdots \le a_{i_k}$ and $1 \le i_1 < i_2 < \cdots < i_k \le n$.

We first consider a feasible solution of (3)-(5). If we throw out all the changed items, the remaining items consist of a well-ordered subsequence.

For any feasible solution x, we can divide the entries of x into two parts, the changed part and the unchanged part. Let the unchanged part be $\{a_{i_1}, a_{i_2}, \cdots, a_{i_k}\}$ with $i_l < i_{l+1}$ for $1 \le l \le k-1$. Then $\{a_{i_1}, a_{i_2}, \cdots, a_{i_k}\}$ is a well-ordered subsequence. The cost of x is equal to $\sum_{i=1}^{n} w_i - \sum_{l=1}^{k} w_{i_l}$. Thus the well-ordered subsequences play a core rule in searching the best solution for the inverse problem.

But well-ordered subsequences may not correspond to feasible solutions because of the bound constraints. Our task now is to find feasible well-ordered subsequences.

For this purpose, let us define a new concept. (i, j) is called a feasible pair if
(a) $1 \le i < j \le n$, and $a_i \le a_j$,
(b) there exist $x_{i+1}, x_{i+2}, \cdots, x_{j-1}$ such that $a_i \le x_{i+1} \le x_{i+2} \le \cdots \le x_{j-1} \le a_j$, and $|x_l - a_l| \le b_l$ for $i+1 \le l \le j-1$.

Using the feasible pairs, we can define a digraph $G_b = (V, A_b)$ such that $V = \{v_i \,|\, 1 \le i \le n\} \cup \{s, t\}$, $A_b = \{(v_i, v_j) \,|\, (i, j)$ is a feasible pair$\} \cup \{(s, v_i) \,|\, 1 \le i \le n\} \cup \{(v_i, t) \,|\, 1 \le i \le n\}$.

Let P be an $s - t$ path in G_b such that $P = \{s, v_{i_1}, v_{i_2}, \cdots, v_{i_k}, t\}$. Then we know that (i_t, i_{t+1}) are feasible pairs for $t = 1, \cdots, k-1$. If there are $\{x_1, x_2, \cdots, x_{i_1 - 1}\}$ and $\{x_{i_k+1}, x_{i_k+2}, \cdots, x_n\}$ such that

$$x_1 \le x_2 \le \cdots \le x_{i_1 - 1} \le a_{i_1}, \text{ and } |x_l - a_l| \le b_l \quad \text{for } 1 \le l \le i_1 - 1, \tag{6}$$

$$a_{i_k} \le x_{i_k+1} \le x_{i_k+2} \le \cdots \le x_n, \text{ and } |x_l - a_l| \le b_t \text{ for } i_k + 1 \le l \le n, \tag{7}$$

then we call such an $s - t$ path a feasible path. For a vertex satisfying (6), we call it a feasible starting vertex, and for a vertex v_i satisfying (7), we call it a feasible ending vertex.

We define lengths of arcs of G_b as follows.

$l(s, v_i) = w_i$, $(s, v_i) \in A_b$ and v_i is a feasible starting vertex;
$l(s, v_i) = -\infty$, $(s, v_i) \in A_b$ and v_i is not a feasible starting vertex;
$l(v_i, t) = 0$, $(v_i, t) \in A_b$ and v_i is a feasible ending vertex;
$l(v_i, t) = -\infty$, $(v_i, t) \in A_b$ and v_i is not a feasible ending vertex;
$l(v_i, v_j) = w_j$, $(v_i, v_j) \in A_b$.

We can easily show that an optimal solution of the inverse sorting problem (3)–(5) corresponds to the longest feasible path from s to t in G_b.

The remaining problems are how to check whether a given pair (i, j) is a feasible one and how to find the longest feasible path. First, we can design the algorithm below to check the feasibility of a given pair.

Checking Algorithm (to check if a pair (i, j) is feasible)

Step 0: Let $x_i = a_i$, $l = i$.

Step 1: If $l = j - 1$, go to Step 4.

Step 2. If $x_l > a_j$, stop and output $False$.

Step 3: If $a_{l+1} + b_{l+1} < x_l$, stop and output $False$. Otherwise set

$$x_{l+1} = \max\{x_l, a_{l+1} - b_{l+1}\}, \tag{8}$$

and $l \leftarrow l + 1$, go to Step 1.

Step 4: If $x_{j-1} \leq a_j$, output $True$, otherwise output $False$.

Consider any two feasible pairs starting from the same i, say (i, j) and (i, k) with $j < k$. Then the sequence $(x_{i+1}, \cdots, x_{j-1})$ generated by the Checking Algorithm for the feasible pair (i, j) will also appear in the subsequence generated by the Checking Algorithm for the feasible pair (i, k). In fact these $x_{i+1}, \cdots, x_{j-1}$ are fully determined by $a_i, \cdots a_{j-1}$ and $b_{i+1}, \cdots, b_{j-1}$. Thus to check the feasibility of (i, k), we do not need to use the checking algorithm from $x_i = a_i$, and it is enough to compute x_l for l from j to $k - 1$. Therefore finding all feasible pairs starting from i can be achieved in $O(n - i)$ time. This idea can be realized by the following greedy algorithm.

Greedy Algorithm (to check pairs (i, k) for all $k > i$)

Step 0: Let $x_i = a_i$.

Step 1: For $l = i + 1$ to $n - 1$, set $x_l = \max\{x_{l-1}, a_l - b_l\}$.

Step 2. For l from $i + 1$ to n, check if $a_i \leq a_l$ and $x_{l-1} \leq a_l$. (i, l) is a feasible pair if yes and infeasible otherwise.

Step 3. Set $l(v_i, v_l) = w_l$ for all such feasible pairs.

Now let us check if a vertex i can be a starting vertex or an ending vertex of a feasible path.

Checking Algorithm for Beginning and Ending Vertices (to check whether i can be a starting/ending vertex of a feasible path)

Step 0: Let $x_i = y_i = a_i$.

Step 1: For $l = i + 1$ to $n - 1$, set $x_l = \max\{x_{l-1}, a_l - b_l\}$, and for $l = i - 1$ to 1, set $y_l = \min\{y_{l+1}, a_l + b_l\}$.

Step 2. If there exists an l within $i + 1 \leq l \leq n$ such that $x_l > a_l + b_l$, then set $l(v_i, t) = -\infty$. Otherwise set $l(v_i, t) = 0$.

Step 3. If there exists an l within $1 \leq l \leq i - 1$, such that $y_l < a_l - b_l$, then set $l(s, v_i) = -\infty$, otherwise set $l(s, v_i) = w_i$.

Running the Greedy Algorithm backward from n to 1, together with the Checking Algorithm for Beginning and Ending Vertices, it is not difficult to see that we can construct the graph G_b in $O(n^2)$ time.

Finally for each vertex v_i in G_b, let $L(i)$ be the length of the longest path from v_i to t. Since G_b is an acyclic graph, it is clear that

$$L(i) = \max\{l(v_i, v_j) + L(j) \,|\, (v_i, v_j) \in A_b\}. \tag{9}$$

If $L(i) = w_j + L(j)$ for $(v_i, v_j) \in A_b$, then we define $\beta(i)$ by the formula $\beta(i) = j$. In other words, $v_{\beta(i)}$ is the next vertex after v_i to realize the longest length $L(i)$.

Now we design a backward algorithm to find the optimal solution for the inverse sorting problem (3) - (5).

Backward Algorithm (to find the longest path in G_b)

Step 0: Let $L(t) = -\infty$, and $i = n$.

Step 1: If $i = 0$, then go to Step 4.

Step 2: Obtain $L(i)$ by (9) and define $\beta(i)$.

Step 3: $i \leftarrow i - 1$, go to Step 1.

Step 4: Set $L(s) = \max\{l(s, v_i) + L(i) \mid (s, v_i) \in A'_b\}$ and let $\beta(s) = i'$ if in the above formula the maximum value is attained at $i = i'$. Stop.

Obviously the $L(s)$ obtained from the above algorithm must be the length of the longest $s - t$ path in G_b. If $L(s) = -\infty$, then the inverse sorting problem has no feasible solution. When a feasible solution exists, the maximum weight well-ordered subsequence can be identified by:

$$i_1 = \beta(s) \rightarrow i_2 = \beta(i_1) \rightarrow i_3 = \beta(i_2) \rightarrow \cdots\cdots i_k,$$

where the last i_k meets the condition that $\beta(i_k) = t$.

Now we consider the complexity of the Backward Algorithm. In Step 2, the computing time is $O(n)$, and in Step 4, the computing time is $O(n)$ too. As these two steps should be executed from $i = n$ to $i = 0$, the total complexity of the Backward Algorithm is $O(n^2)$.

Since the construction of G_b can be achieved in $O(n^2)$ time, and the complexity of the Backward Algorithm is $O(n^2)$ time too, the inverse sorting problem under Hamming distance can be solved in $O(n^2)$ time.

3 Partial Inverse Sorting Problems

3.1 Partial Inverse Sorting Problem with Free Positions

In this subsection, we consider the partial inverse sorting problem with free positions, i.e. the problem (PIS-1) stated in Section 1. We show that the problem can be solved by a combination of an inverse sorting problem and a sorting problem. Let π be the given permutation over the subset J of I and (a^*, ω) be an optimal solution to the problem (PIS-1) under some separable convex cost function $C(x - a)$. We denote by $a^*|_J$ the part of a^* restricted on the subset J. Then $a^*|_J$ is a feasible solution of the inverse sorting problem restricted on J as follows:

$$\min C(x|_J - a|_J) = \sum_{i \in J} c_i(x_i - a_i) \tag{10}$$

$$\text{s.t. } x_{\pi(1)} \leq x_{\pi(2)} \leq \cdots \leq x_{\pi(|J|)}. \tag{11}$$

Denote by C^{opt} and $C^{\text{opt}}|_J$ respectively the minimum value (cost) of the problem (PIS-1) and the minimum value of the restricted inverse sorting problem defined by (10) and (11). It is obvious that

$$C^{\text{opt}} = \sum_{i=1}^{n} c_i(a_i^* - a_i) \geq \sum_{i \in J} c_i(a_i^* - a_i) \geq C^{\text{opt}}|_J.$$

Therefore, $C^{\text{opt}}|_J$ is a lower bound of C^{opt}.

Now let $x^*|_J$ be the optimal solution of the restricted inverse sorting problem (10)-(11). We define

$$b_i = x_i^*, \qquad i \in J; \tag{12}$$

$$b_i = a_i, \qquad i \in \bar{J} = I \setminus J, \tag{13}$$

and consider the sorting problem of arranging $\{b_1, b_2, \cdots\cdots b_n\}$ in a non-decreasing order. Let ω be the permutation of this sorting problem. From (11) and (12), we know that

$$b_{\pi(1)} \leq b_{\pi(2)} \leq \cdots \leq b_{\pi(|J|)},$$

that is, ω covers π. So, (b, ω) is a feasible solution of the partial inverse sorting problem with free positions (PIS-1). Moreover, the objective value of (PIS-1) under this feasible solution is

$$C(b-a) = \sum_{i=1}^{n} c_i(b_i - a_i) = \sum_{i \in J} c_i(x_i^* - a_i) = C^{\text{opt}}|_J,$$

which reaches the lower bound of C^{opt}. Therefore, $C(b-a) = C^{\text{opt}}$ and (b, ω) is the optimal solution to the problem (PIS-1).

To summarize, the partial inverse sorting problem with free positions can be solved by two steps. In the first step, we solve a restricted inverse sorting problem (10)-(11); and in the second step, we solve a full size sorting problem (12)-(13).

3.2 Partial Inverse Sorting Problem with Fixed Positions

Now we consider the partial inverse sorting problem with fixed positions. i.e. the problem (PIS-2) stated in Section 1. In this section, We consider only the three simplest cost functions. They are l_2, l_∞ and l_1 norms of the deviation, i.e., $C_2(x-a) = \sum_{i \in I}(x_i - a_i)^2$, $C_\infty(x-a) = \max\{|x_i - a_i| \ : \ i \in I\}$ and $C_1(x-a) = \sum_{i \in I} |x_i - a_i|$. We show that the partial inverse problem under these three cost functions can be solved by a combination of a sorting problem and an inverse sorting problem

Let π be a given permutation over the subset J of I and the positions of numbers in J are fixed in I. Let $\bar{J} = I \setminus J$. We claim that

Claim *For the three cost functions above, there exists an optimal solution* (a^*, ω) *for the problem (PIS-2) such that*

$$a_i^* \le a_j^* \quad \text{if} \quad a_i \le a_j \quad \text{for} \quad i, j \in \bar{J}. \tag{14}$$

That is $\omega|_{\bar{J}}$ is in the non-decreasing order of $\{a_i : i \in \bar{J}\}$. Where $\omega|_{\bar{J}}$ is the permutation ω restricted on $\bar{J}$.

In fact, suppose that there exist $i, j \in \bar{J}$ such that $a_i^* \le a_j^*$ but $a_i > a_j$. Suppose that $\omega(i') = i$ and $\omega(j') = j$ with $i' < j'$.

Let us construct a new solution (b, ϖ) such that

$$\varpi(k) = \omega(k) \qquad k \ne i', j' \tag{15}$$
$$\varpi(i') = \omega(j') = j \tag{16}$$
$$\varpi(j') = \omega(i') = i \tag{17}$$
$$b_{\varpi(k)} = a^*_{\omega(k)} \qquad k \ne i', j' \tag{18}$$
$$b_{\varpi(i')} = a^*_{\omega(i')} = a_i^* \tag{19}$$
$$b_{\varpi(j')} = a^*_{\omega(j')} = a_j^* \tag{20}$$

That is, the new solution is gotten by interchanging the positions of numbers at positions i' and j' in ω, but keeping the positions of other numbers within ω unchanged, and keeping the modified number unchanged at each position.

Let us investigate the changes of the costs. For the l_2 norm cost function, the original cost of modifying numbers a_i and a_j is $(a_i^* - a_i)^2 + (a_j^* - a_j)^2$, and the corresponding new cost is $(b_j - a_j)^2 + (b_i - a_i)^2 = (a_i^* - a_j)^2 + (a_j^* - a_i)^2$. Notice that $a_i^* \le a_j^*$ and $a_i > a_j$. So, $(a_i - a_j)(a_i^* - a_j^*) \le 0$. Using this inequality, we can easily deduce that

$$(a_i^* - a_i)^2 + (a_j^* - a_j)^2 \ge (a_i^* - a_j)^2 + (a_j^* - a_i)^2 = (b_j - a_j)^2 + (b_i - a_i)^2.$$

Hence the interchange may reduce but never increase the total modification cost.

By enumerating all possible cases, we can also show that $C_\infty(a^* - a) \ge C_\infty(b - a)$ and $C_1(a^* - a) \ge C_1(b - a)$. That is, the Claim is true under both l_∞ and l_1 norm functions.

By this claim, we can sort the numbers in $\bar{J}$ to obtain a partial order π' first. Then we construct a permutation ω by combining π' and π. Since the positions of numbers in J are fixed in ω, both π' and π are known, and hence ω is well-defined. Then we solve an inverse sorting problem as follows

$$\min C_\infty(x - a) \quad (\text{or} \quad C_1(x - a), \quad C_2(x - a)) \tag{21}$$
$$\text{s.t. } x_{\omega(1)} \le x_{\omega(2)} \le \cdots \le x_{\omega(n)} \tag{22}$$

The optimal solution x^* of (21) and (22) together with ω form an optimal solution (x^*, ω) of (PIS-2) under l_2, l_∞ and l_1 norm functions.

Since the complexity of sorting at the first stage is $O((n - |J|)\ln((n - |J|)))$, and the complexity of inverse sorting at the second stage is $O(n)$, $O(n)$ and

$O(n\ln(n))$ with respect to l_2, l_∞ and l_1 cost functions respectively, the total complexity for (PIS-2) under l_2, l_∞ and l_1 norm functions is at most $O(n\ln(n))$.

We would like to note that solving the partial inverse sorting problem with fixed positions under l_2, l_∞ and l_1 norm cost functions can be divided into two steps too. Interestingly, in contrast to the procedure for solving (PIS-1), the first step to solve (PIS-2) is to solve a restricted version of a sorting problem, and the second step is to solve a full size inverse sorting problem. That is, the order and size of such two subproblems are reversed.

4 Concluding Remarks

In this paper, we first consider an inverse sorting problem under weighted Hamming distance with bound constraints and show that this inverse problem can be solved in $O(n^2)$ time. Then we consider two partial inverse sorting problems, the partial inverse sorting problem with free positions and the partial inverse sorting problem with fixed positions and show that both problems can be solved under some special cost functions. There are many unsolved problems about the partial inverse sorting problems. For example, for the partial inverse sorting problem with fixed positions under *weighted* l_1, l_∞ and l_2 norm cost functions, the partial inverse sorting problem with fixed positions under l_1, l_∞ and l_2 norm cost functions but *with bound constraints on changes*, and both types of partial inverse sorting problems *under Hamming distance*, it is so far unknown whether there are polynomial algorithms. A further effort is needed.

References

1. R.K. Ahuja, *Inverse optimization*, Slides of talk given at Georgia Institute of Technology, Atlanta, GA; November 1998, downloadable from http://www.ise.ufl.edu/ahuja/
2. R.K. Ahuja, and J.B. Orlin, *A fast scaling algorithm for minimizing separable convex functions subject to chain constraints*, Operations Research, 49, 2001, 784-789.
3. Y. He, B.W. Zhang, and E.Y. Yao, *Weighted inverse minimum spanning tree problems under Hamming distance*, Journal of Combinatorial Optimization, 9, 2005, 91-100.
4. B.W. Zhang, J.Z. Zhang, and Y. He, *The center location improvement problem under the Hamming distance*, Journal of Combinatorial Optimization, to appear.

Author Index