LNCS 6831

Weifan Wang
Xuding Zhu
Ding-Zhu Du (Eds.)

Combinatorial Optimization and Applications

5th International Conference, COCOA 2011
Zhangjiajie, China, August 2011
Proceedings

Springer

Weifan Wang Xuding Zhu Ding-Zhu Du (Eds.)

Combinatorial Optimization and Applications

5th International Conference, COCOA 2011
Zhangjiajie, China, August 4-6, 2011
Proceedings

Springer

Volume Editors

Weifan Wang
Xuding Zhu
Zhejiang Normal University
688 Yingbin Road, Jinhua, Zhejiang Province, 321004 China
E-mail: wwf@zjnu.cn, xudingzhu@gmail.com

Ding-Zhu Du
University of Texas at Dallas, Department of Computer Science
Richardson, TX 75080, USA
E-mail: dzdu@utdallas.edu

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-22615-1 e-ISBN 978-3-642-22616-8
DOI 10.1007/978-3-642-22616-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011931984

CR Subject Classification (1998): F.2, C.2, G.2-3, I.3.5, G.1.6

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The 5th Annual International Conference on Combinatorial Optimization and Applications, COCOA 2011, took place in Zhangjiajie, China, August 4-6, 2011. Past COCOA conferences were held in Xi'an, China (2007), Newfoundland, Canada (2008), Huangshan, China (2009) and Hawaii, USA (2010).

COCOA 2011 provided a forum for researchers working in the areas of combinatorial optimization and its applications. In addition to theoretical results, the conference is particularly focused on recent works on experimental and applied research of general algorithmic interest. The Program Committee received submissions from various countries and regions over the world. From 65 submissions, 30 papers were selected for presentation at the conference. In addition, the conference organized two invited sessions exploring recent developments of optimizations in graphs and network technology, especially network optimizations.

We wish to thank the authors for submitting their papers to the conference. We are grateful to the members of the Program Committee and the external referees for their work within demanding time constraints. We thank the Organizing Committee for their contribution to making the conference a success. We also thank Lidong Wu and Jiaofei Zhong for helping us create and update the conference website and maintaining the Springer Online Conference Service system. We also wish to thank all assistants and workers who served this conference at Qinghe Jin Jiang International Hotel, Zhangjiajie City, Hunan Province, China. Without their service, this conference would not have been successful.

Finally, we thank the conference sponsors and supporting organizations for their support and assistance. COCOA 2011 was supported in part by the National Natural Science Foundation of China, the Chinese Academy of Sciences, Zhejiang Normal University, and the University of Texas at Dallas.

August 2011

Weifan Wang
Xuding Zhu
Ding-Zhu Du

Organization

COCOA 2010 was organized by the Institute of Applied Mathematics, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, and the Department of Computer Science, the University of Texas at Dallas, in cooperation with the Zhejiang Normal University.

Executive Committee

General Co-chairs	Yuehua Bu (Zhejiang Normal University, China)
	Xiaodong Hu (Chinese Academy of Sciences, China)
	Weili Wu (University of Texas at Dallas, USA)
PC Co-chairs	Weifan Wang (Zhejiang Normal University, China)
	Xuding Zhu (Zhejiang Normal University, China)
	Ding-Zhu Du (University of Texas at Dallas, USA)
Publicity Co-chairs	Jiaofei Zhong (University of Texas at Dallas, USA)
	Lidong Wu (University of Texas at Dallas, USA)

Program Committee

Wolfgang Bein	University of Nevada, USA
Xujin Chen	Chinese Academy of Sciences, China
Yongxi Cheng	Xi'an Jiaotong University, China
Bhadrachalam Chitturi	Amrita Vishwa Vidyapeetham University, India
Ovidiu Daescu	University of Texas at Dallas, USA
Bhaskar Dasgupta	University of Illinois at Chicago, USA
Hongwei (David) Du	Harbin Institute of Technology, China
Zhenhua Duan	Xidian University, China
Omer Egecioglu	University of California, Santa Barbara, USA
Xiaofeng Gao	Georgia Gwinnett College, USA
Juraj Hromkovic	ETH Zentrum, Switzerland
Wenlian Hsu	Academia Sinica, Taiwan
Kazuo Iwama	Kyoto University, Japan
Liying Kang	Shanghai University, China
Donghyun (David) Kim	North Carolina Central University, USA
Minming Li	City University of Hong Kong, China
Mitsunori Ogihara	University of Miami, USA
Suneeta Ramaswami	Rutgers University, USA
Xiaoming Sun	Tsinghua University, China

Table of Contents

The Complexity of Testing Monomials in Multivariate Polynomials

Zhixiang Chen and Bin Fu

Department of Computer Science,
University of Texas-Pan American,
Edinburg, TX 78539, USA
{chen,binfu}@cs.panam.edu

Abstract. The work in this paper is to initiate a theory of testing monomials in multivariate polynomials. The central question is to ask whether a polynomial represented by certain economically compact structure has a multilinear monomial in its sum-product expansion. The complexity aspects of this problem and its variants are investigated with two objectives. One is to understand how this problem relates to critical problems in complexity, and if so to what extent. The other is to exploit possibilities of applying algebraic properties of polynomials to the study of those problems. A series of results about $\Pi\Sigma\Pi$ and $\Pi\Sigma$ polynomials are obtained in this paper, laying a basis for further study along this line.

Keywords: Multivariate polynomials, multilinear monomials, monomial testing, algebra, complexity.

1 Introduction

We begin with two examples to exhibit the motivation and necessity of the study about the monomial testing problem for multivariate polynomials. The first is about testing a k-path in any given undirected graph $G = (V, E)$ with $|V| = n$, and the second is about the satisfiability problem. Throughout this paper, polynomials refer to those with multiple variables.

For any fixed integer $c \geq 1$, for each vertex $v_i \in V$, define a polynomial $p_{k,i}$ as follows:

$$p_{1,i} = x_i^c,$$
$$p_{k+1,i} = x_i^c \left(\sum_{(v_i,v_j)\in E} p_{k,j} \right), \ k > 1.$$

We define a polynomial for G as

$$p(G,k) = \sum_{i=1}^{n} p_{k,i}.$$

Obviously, $p(G,k)$ can be represented by an arithmetic circuit. It is easy to see that the graph G has a k-path $v_{i_1} \cdots v_{i_k}$ iff $p(G,k)$ has a monomial of $x_{i_1}^c \cdots x_{i_k}^c$ of degree ck in its sum-product expansion. G has a Hamiltonian path iff $p(G,n)$ has the monomial $x_1^c \cdots x_n^c$ of degree cn in its sum-product expansion. One can also see that a path with some loop can be characterized by a monomial as well. Those observations show that testing monomials in polynomials is closely related to solving k-path, Hamiltonian path and other problems about graphs. When $c = 1$, $x_{i_1} \cdots x_{i_k}$ is multilinear. The problem of testing multilinear monomials has recently been exploited by Koutis [13] and Williams [18] to design innovative randomized parameterized algorithms for the k-path problem.

Now, consider any CNF formula $f = f_1 \wedge \cdots \wedge f_m$, a conjunction of m clauses with each clause f_i being a disjunction of some variables or negated ones. We may view conjunction as multiplication and disjunction as addition, so f looks like a *"polynomial"*, denoted by $p(f)$. $p(f)$ has a much simpler $\Pi\Sigma$ representation, as will be defined in the next section, than general arithmetic circuits. Each *"monomial"* $\pi = \pi_1 \ldots \pi_m$ in the sum-product expansion of $p(f)$ has a literal π_i from the clause f_i. Notice that a boolean variable $x \in Z_2$ has two properties of $x^2 = x$ and $x\bar{x} = 0$. If we could realize these properties for $p(f)$ without unfolding it into its sum-product, then $p(f)$ would be a *"real polynomial"* with two characteristics: (1) If f is satisfiable then $p(f)$ has a multilinear monomial, and (2) if f is not satisfiable then $p(f)$ is identical to zero. These would give us two approaches towards testing the satisfiability of f. The first is to test multilinear monomials in $p(f)$, while the second is to test the zero identity of $p(f)$. However, the task of realizing these two properties with some algebra to help transform f into a needed polynomial $p(f)$ seems, if not impossible, not easy. Techniques like arithmetization in Shamir [17] may not be suitable in this situation. In many cases, we would like to move from Z_2 to some larger algebra so that we can enjoy more freedom to use techniques that may not be available when the domain is too constrained. The algebraic approach within $Z_2[Z_2^k]$ in Koutis [13] and Williams [18] is one example along the above line. It was proved in Bshouty *et al.* [5] that extensions of DNF formulas over Z_2^n to Z_N-DNF formulas over the ring Z_N^n are learnable by a randomized algorithm with equivalence queries, when N is large enough. This is possible because a larger domain may allow more room to utilize randomization.

There has been a long history in complexity theory with heavy involvement of studies and applications of polynomials. Most notably, low degree polynomial testing/representing and polynomial identity testing have played invaluable roles in many major breakthroughs in complexity theory. For example, low degree polynomial testing is involved in the proof of the PCP Theorem, the cornerstone of the theory of computational hardness of approximation and the culmination of a long line of research on IP and PCP (see, Arora *at el.* [2] and Feige *et al.* [8]). Polynomial identity testing has been extensively studied due to its role in various aspects of theoretical computer science (see, for example, Kabanets and Impagliazzo [11]) and its applications in various fundamental results such as Shamir's IP=PSPACE [17] and the AKS Primality Testing [1]. Low degree polynomial representing [14] has been sought for so as to prove important results in circuit complexity, complexity class separation and subexponential time learning of boolean functions (see, for examples, Beigel [4], Fu[9] and Klivans and Servedio [12]). These are just a few examples. A survey of the related literature is certainly beyond the scope of this paper.

The above two examples of the k-path testing and satisfiability problems, the rich literature about polynomial testing and many other observations have motivated us to develop a new theory of testing monomials in polynomials represented by economically compact structures. The monomial testing problem is related to, and somehow complements with, the low degree testing and the identity testing of polynomials. We want to investigate various complexity aspects of the monomial testing problem and its variants with two folds of objectives. One is to understand how this problem relates to critical problems in complexity, and if so to what extent. The other is to exploit possibilities of applying algebraic properties of polynomials to the study of those critical problems.

The paper is organized as follows. We first define $\Pi\Sigma\Pi$ and $\Pi\Sigma$ polynomials. The first is a product of clauses such that each clause is a sum of terms and each term is a product of variables. The second is like the first except that each term is just one variable. These polynomials have easy depth-3 or depth-2 circuit representations that have been extensively studied for the polynomial identity testing problem. We prove a series of results: The multilinear monomial testing problem for $\Pi\Sigma\Pi$ polynomials is NP-hard, even when each clause has at most three terms. The testing problem for $\Pi\Sigma$ polynomials is in P, and so is the testing for two-term $\Pi\Sigma\Pi$ polynomials. However, the testing for a product of one two-term $\Pi\Sigma\Pi$ polynomial and another $\Pi\Sigma$ polynomial is NP-hard. We also prove that testing c-monomials for two-term $\Pi\Sigma\Pi$ polynomials is NP-hard for any $c > 2$, but the same testing is in P for $\Pi\Sigma$ polynomials. Finally, two parameterized algorithms was devised for three-term $\Pi\Sigma\Pi$ polynomials and products of two-term $\Pi\Sigma\Pi$ and $\Pi\Sigma$ polynomials. These results have laid a basis for further study about testing monomials.

2 Notations and Definitions

Let $\mathcal{P}$ be a field. For variables $x_1, \ldots, x_n$, let $\mathcal{P}[x_1, \cdots, x_n]$ denote the communicative ring of all the n-variate polynomials with coefficients from $\mathcal{P}$. For $1 \leq i_1 < \cdots < i_k \leq n$, $\pi = x_{i_1}^{j_1} \cdots x_{i_k}^{j_k}$ is called a monomial. The degree of π, denoted by $\deg(\pi)$, is $\sum_{s=1}^{k} j_s$. π is multilinear, if $j_1 = \cdots = j_k = 1$, i.e., π is linear in all its variables $x_{i_1}, \ldots, x_{i_k}$. For any given integer $c \geq 1$, π is called a c-monomial, if $1 \leq j_1, \ldots, j_k < c$.

An arithmetic circuit, or circuit for short, is a direct acyclic graph with $+$ gates of unbounded fan-ins, $\times$ gates of two fan-ins, and all terminals corresponding to variables. The size, denoted by $s(n)$, of a circuit with n variables is the number of gates in it. A circuit is called a formula, if the fan-out of every gate is at most one, i.e., the underlying direct acyclic graph is a tree.

By definition, any polynomial $p(x_1, \dots, x_n)$ can be expressed as a sum of a list of monomials, called the sum-product expansion. The degree of the polynomial is the largest degree of its monomials in the expansion. With this expression, it is trivial to see whether $p(x_1, \dots, x_n)$ has a multilinear monomial, or a monomial with any given pattern. Unfortunately, this expression is essentially problematic and infeasible to realize, because a polynomial may often have exponentially many monomials in its expansion.

In general, a polynomial $p(x_1, \dots, x_n)$ can be represented by a circuit or some even simpler structure as defined in the following. This type of representation is simple and compact and may have a substantially smaller size, say, polynomially in n, in comparison with the number of all monomials in the sum-product expansion. The challenge is how to test whether $p(x_1, \dots, x_n)$ has a multilinear monomial or some needed monomial, efficiently without unfolding it into its sum-product expansion?

Definition 1. *Let $p(x_1, \dots, x_n) \in \mathcal{P}[x_1, \dots, x_n]$ be any given polynomial. Let $m, s, t \geq 1$ be integers.*

- *$p(x_1, \dots, x_n)$ is said to be a $\Pi_m\Sigma_s\Pi_t$ polynomial, if $p(x_1, \dots, x_n) = \prod_{i=1}^{t} F_i$, $F_i = \sum_{j=1}^{r_i} X_{ij}$ and $1 \leq r_i \leq s$, and $deg(X_{ij}) \leq t$. We call each F_i a clause. Note that X_{ij} is not a monomial in the sum-product expansion of $p(x_1, \dots, x_n)$ unless $m = 1$. To differentiate this subtlety, we call X_{ij} a term.*
- *In particular, we say that $p(x_1, \dots, x_n)$ is a $\Pi_m\Sigma_s$ polynomial, if it is a $\Pi_m\Sigma_s\Pi_1$ polynomial. Here, each clause is a linear addition of single variables. In other word, each term has degree 1.*
- *When no confusing arises from the context, we use $\Pi\Sigma\Pi$ and $\Pi\Sigma$ to stand for $\Pi_m\Sigma_s\Pi_t$ and $\Pi_m\Sigma_s$ respectively.*
  Similarly, we use $\Pi\Sigma_s\Pi$ and $\Pi\Sigma_s$ to stand for $\Pi_m\Sigma_s\Pi_t$ and $\Pi_m\Sigma_s$ respectively, emphasizing that every clause in a polynomial has at most s terms or is a linear addition of at most s single variables.
- *For any given integer $k \geq 1$, $p(x_1, \dots, x_n)$ is called a k-$\Pi\Sigma\Pi$ polynomial, if each of its terms has k distinct variables.*
- *$p(x_1, \dots, x_n)$ is called a $\Pi\Sigma\Pi \times \Pi\Sigma$ polynomial, if $p(x_1, \dots, x_n) = p_1p_2$ such that p_1 is a $\Pi\Sigma\Pi$ polynomial and p_2 is a $\Pi\Sigma$ polynomial. Similarly, $p(x_1, \dots, x_n)$ is called a k-$\Pi\Sigma\Pi \times \Pi\Sigma$ polynomial, if $p(x_1, \dots, x_n) = p_1p_2$ such that p_1 is a k-$\Pi\Sigma\Pi$ polynomial and p_2 is a $\Pi\Sigma$ polynomial.*

It is easy to see that a $\Pi_m\Sigma_s\Pi_t$ or $\Pi_m\Sigma_s$ polynomial may has as many as s^m monomials in its sum-product expansion.

On the surface, a $\Pi_m\Sigma_s\Pi_t$ polynomial *"resembles"* a SAT formula, especially when $t = 1$. Likewise, a $\Pi_m\Sigma_3\Pi_t$ ($\Pi_m\Sigma_2\Pi_t$) polynomial *"resembles"* a 3SAT (2SAT) formula, especially when $t = 1$. However, negated variables are not involved in a polynomials. Furthermore, as pointed out in the previous section, it is not easy, if not impossible, to have some easy algebra to deal with the properties of $x^2 = x$ and $x \cdot \bar{x} = 0$ in a field, especially when the field is larger than Z_2. Also, as pointed out before, the arithmetization technique in Shamir [17] is not applicable to this case.

Throughout the rest of the paper, we shall focus on nonnegative integer coefficients in polynomials.

3 $\Pi\Sigma\Pi$ Polynomials

Given any $\Pi_m\Sigma_s\Pi_t$ polynomial $p(x_1, \ldots, x_n) = p_1 \cdots p_m$, one can nondeterministically choose a term π_i from the clause p_i and then check whether $\pi_1 \cdots \pi_m$ is a multilinear monomial. So the problem of testing multilinear monomials in a $\Pi\Sigma\Pi$ polynomial is in NP. In the following we show that this problem is also NP-hard.

Theorem 1. *It is NP-hard to test whether a* 2-$\Pi_m\Sigma_3\Pi_2$ *polynomial has a multilinear monomial in its sum-product expansion.*

Note that every clause in such a 2-$\Pi_m\Sigma_3\Pi_2$ polynomial has at most three terms such that each term has at most two distinct variables.

Proof. We reduce 3SAT to the given problem. Let $f = f_1 \wedge \cdots \wedge f_m$ be a 3SAT formula. Without loss of generality, we assume that every variable x_i in f appears at most three times, and if x_i appears three times, then x_i itself occurs twice and $\bar{x}_i$ once. (It is easy to see that a simple preprocessing procedure can transform any 3SAT formula to satisfy these properties.)

Let x_i be any given variable in f, we introduce new variables to replace it. If x_i appears only once then we replace the appearance of x_i (or $\bar{x}_i$) by a new variable y_{i1}. When x_i appears twice, then we do the following: If x_i (or its negation $\bar{x}_i$) occurs twice, then replace the first occurrence by a new variable y_{i1} and the second by y_{i2}. If both x_i and $\bar{x}_i$ occur, then replace both occurrences by y_{i1}. When x_i occurs three times with x_i appearing twice and $\bar{x}_i$ once, then replace the first x_i by y_{i1} and the second by y_{i2}, and replace $\bar{x}_i$ by $y_{i1}y_{i2}$. This procedure of replacing all variables in f, negated or not, with new variables can be carried out easily in quadratic time.

Let $p = p_1 \cdots p_m$ be polynomial resulting from the above replacement process. Here, p_i corresponds to f_i with boolean literals being replaced. Clearly, p is a 2-$\Pi_m\Sigma_3\Pi_2$ polynomial.

We now consider the sum-product expansion of $f = f_1 \cdots f_m$. It is easy to see that f is satisfiable iff its sum-product expansion has a product

$$\psi = \tilde{x}_{i_1} \cdots \tilde{x}_{i_m},$$

where the literal $\tilde{x}_{i_j}$ is from the clause f_j and is either x_{i_j} or $\bar{x}_{i_j}$, $1 \leq j \leq m$. Furthermore, the negation of $\tilde{x}_{i_j}$ must not occur in π.

Let $t(\tilde{x}_{i_j})$ denote the replacement of $\tilde{x}_{i_j}$ by new variables y_{i_j1} and/or y_{i_j2} as described above to transform f to p. Then, $t(\tilde{x}_{i_j})$ is a term in the clause p_j. Hence,

$$t(\psi) = t(\tilde{x}_{i_j}) \cdots t(\tilde{x}_{i_m})$$

is a monomial in the sum-product expansion of p. Moreover, $t(\psi)$ is multilinear, because a variable and its negation cannot appear in ψ at the same time.

On the other hand, assume that

$$\pi = \pi_1 \cdots \pi_m$$

is a multilinear monomial in p with the term π_{i_j} in the clause p_j. Let $t^{-1}(\cdot)$ denote the reversal replacement of $t(\cdot)$. Then, by the procedure of the replacement above, $t^{-1}(\pi_{i_j})$ is a variable or the negation of a variable in f_j. Thus,

$$t^{-1}(\pi) = t^{-1}(\pi_1) \cdots t^{-1}(\pi_m)$$

is a product in the sum-product expansion of f. Since π is multilinear, a variable and its negation cannot appear in $t^{-1}(\pi)$ at the same time. This implies that f is satisfiable by an assignment of setting all the literals in $t^{-1}(\pi)$ true.

We give an example to illustrate the variable replacement procedure given in the above proof. Given a 3SAT formula

$$f = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_4 \vee x_5),$$

the polynomial for f after variable replacements is

$$p(f) = (y_{11} + y_{21}y_{22} + y_{31})(y_{11}y_{12} + y_{21} + y_{41})(y_{12} + y_{22} + y_{31})(y_{42} + y_{51}).$$

The truth assignment satisfying f as determined by the product $x_3 \cdot \bar{x}_1 \cdot x_2 \cdot x_4$ is one to one correspondent to the multilinear monomial $y_{31} \cdot y_{11}y_{12} \cdot y_{22} \cdot y_{42}$ in $p(f)$.

Two corollaries follow immediately from this theorem.

Corollary 1. *For any $s \geq 3$, it is NP-hard to test whether a $\Pi_m\Sigma_s\Pi_t$ polynomial has multilinear monomials in its sum-product expansion.*

Corollary 2. *It is NP-hard to test whether a polynomial has multilinear monomials in its sum-product expansion, when the polynomial is represented by a general arithmetic circuit.*

The NP-hardness in Corollary 2 was obtained by Koutis [13].

4 $\Pi\Sigma$ Polynomials

Note that every clause in a $\Pi\Sigma$ polynomial p is a linear addition of single variables. p looks very much like a SAT formula. But this kind of structural "resemblance" is very superficial, as we will show in the following that the multilinear monomial testing problem for p is in P. This shows that terms with single variables do not have the same expression power as boolean variables and their negations together can achieve. As exhibited in the proof of Theorem 1, terms with two variables are equally powerful as boolean variables together with their negations. Hence, it is interesting to see that a complexity boundary exists between polynomials with terms of degree 1 and those with terms of degree 2.

Theorem 2. *There is a $O(ms\sqrt{m+n})$ time algorithm to test if a $\Pi_m\Sigma_s$ polynomial has a multilinear monomial in its sum-product expansion.*

Proof. Let $f(x_1,\dots,x_n) = f_1\dots f_m$ be any given $\Pi_m\Sigma_s$ polynomial. Without loss of generality, we assume that each clause has exactly s many terms, i.e., $f_i = \sum_{j=1}^{s} x_{ij}$, $1 \le i \le s$. We shall reduce the problem of testing multilinear monomials in $f(x_1,\dots,x_n)$ to the problem of finding a maximum matching in some bipartite graph.

We construct a bipartite graph $G = (V_1 \cup V_2, E)$ as follows. $V_1 = \{v_1,\dots,v_m\}$ so that each v_i represents the clause f_i. $V_2 = \{x_1,\dots,x_n\}$. For each clause f_i, if it contains a variable x_j then we add an edge (v_i, x_j) into E.

Suppose that $f(x_1,\dots,x_n)$ has a multilinear monomial

$$\pi = x_{i_1}\cdots x_{i_m}$$

with x_{i_j} in f_j, $1 \le j \le m$. Then, all the variables in π are distinct. Thus, we have a maximum matching of size m

$$(v_1, x_{i_1}),\dots,(v_m, x_{i_m}).$$

Now, assume that we have a maximum matching of size m

$$(v_1, x'_{i_1}),\dots,(v_1, x'_{i_m}).$$

Then, all the variables in the matching are distinct. Moreover, by the construction of the graph G, x'_{i_j} are in the clause f_j, $1 \le j \le m$. Hence,

$$\pi' = x'_{i_1}\cdots x'_{i_m}$$

is a multilinear monomial in $f(x_1,\dots,x_n)$

It is well-known that finding a maximum matching in a bipartite graph can be done in $O(|E|\sqrt{|V|})$ time [3]. So the above reduction shows that we can test whether $f(x_1,\dots,x_n)$ has a multilinear monomial in $O(ms\sqrt{m+n})$ time, since the graph G has $m+n$ vertices and at most ms edges.

In the following, we give an extension of Theorem 2.

Theorem 3. *There is a $O(tc^k ms\sqrt{m+n})$ time algorithm to test whether any given $\Pi_k\Sigma_c\Pi_t \times \Pi_m\Sigma_s$ polynomial has a multilinear monomial in its sum-product expansion.*

Proof. Let $p = p_1p_2$ be any given $\Pi_k\Sigma_c\Pi_t \times \Pi_m\Sigma_s$ polynomial such that $p_1 = f_1\cdots f_k$ is a $\Pi_k\Sigma_c\Pi_t$ polynomial and $p_2 = g_1\cdots g_m$ is a $\Pi_m\Sigma_s$ polynomial. Note that every clause f_i in p_1 has at most c terms with degree at most t. So, p_1 has at most c^k products in its sum-product expansion. Hence, in $O(tc^k)$ time, we can list all the products in that expansion, and let $\mathcal{C}$ denote the set of all those products.

It is obvious that p has a multilinear monomial, iff there is one product $\psi \in \mathcal{C}$ such that the polynomial ψp_2 has a multilinear monomial.

Now, for any product $\psi \in \mathcal{C}$, we consider how to test whether the polynomial

$$p(\psi) = \psi \cdot p_2 = \psi \cdot g_1 \cdots g_m$$

have a multilinear polynomial. Let

$$\pi = \psi \cdot \pi_1 \cdots \pi_m$$

be an arbitrary product in the sum-product expansion of $p(\psi)$ with the term π_i in g_i, $1 \leq i \leq m$. Since ψ is fixed, in order to make π to be multilinear, each π_i must not have a variable in ψ. This observation helps us devise a one-pass *"purging"* process to eliminate all the variables in every clause of g_i that cannot be included in a multilinear monomial in $p(\psi)$. The purging works as follows: For each clause g_i, eliminate all its variables that also appear in ψ. Let g'_i be the resulting clause of g_i, and $p'_2 = g'_1 \cdots g'_m$ be the resulting polynomial of p_2. If any g'_i is empty, then there is no multilinear monomials in $\psi \cdot p'_2$, hence no multilinear monomials in $p(\psi)$. Otherwise, by Theorem 2, we can decide whether p'_2 has a multilinear monomial, hence whether $p(\psi)$ has a multilinear monomial, in $O(ms\sqrt{m+n})$ time.

Putting all the steps together, we can test whether p has a multilinear monomial in $O(tc^k ms\sqrt{m+n})$ time.

5 $\Pi\Sigma_2\Pi$ Polynomials

In Section 3, we has proved that the multilinear monomial testing problem for any $\Pi\Sigma_s\Pi$ polynomials with at most $s \geq 3$ terms in each clause is NP-hard. In this section, we shall show that another complexity boundary exists between $\Pi\Sigma_3\Pi$ polynomials and $\Pi\Sigma_2\Pi$ polynomials. As noted before, a $\Pi\Sigma_2\Pi$ polynomial may look like a 2SAT formula, but they are essentially different from each other. For example, unlike 2SAT formulas, no implication can be derived for two terms in a clause. Thus, the classical algorithm based on implication graphs for 2SAT formulas by Aspvall, Plass and Tarjan [3] does not apply to $\Pi\Sigma_2\Pi$ polynomials. The implication graphs can also help prove that 2SAT is NL-complete [15]. But we do not know whether the monomial testing problem for $\Pi\Sigma_2\Pi$ polynomials is NL-complete or not. We feel that it may be not. There is another algorithm for solving 2SAT in quadratic time via repeatedly *"purging"* contradicting literals. The algorithm devised in the following more or less follows a similar approach of that quadratic time algorithm.

Theorem 4. *There is a quadratic time algorithm to test whether any given $\Pi_m\Sigma_2\Pi_t$ polynomial has a multilinear monomial in its sum-product expansion.*

Proof. Let $f = f_1 \cdots f_m$ be any given $\Pi_m\Sigma_2\Pi_t$ polynomial such that $f_i = (T_{i1} + T_{i2})$ and each term has degree at most t. Let

$$\pi = \pi_1 \cdots \pi_m$$

be any monomial in the sum-product expansion of f. Here term π_i is either T_{i1} or T_{i2}, $1 \leq i \leq m$. Observe that π is multilinear, iff any two terms in it must not share a common variable. We now devise a *"purging"* based algorithm to decide whether a multilinear monomial π exists in f. The purging part of this algorithm is similar to what is used in the proof of Theorem 3.

The purging algorithm works as follows. We select any clause f_i from f, and choose a term in f_i for π_i. we purge all the terms in the remaining clauses that share a common variable with π_i. Once we find one clause with one term being purged but with the other left, we then choose this remaining term in that clause to repeat the purging process.

The purging stops for π_i when one of the three possible scenarios happens:

(1) We find one clause f_j with two terms being purged. In this case, any of the two terms in f_j cannot be chosen to form a multilinear monomial along with π_i. So, we have to choose the other term in f_i for π, if that term has not been chosen. We use this π_i to repeat the same purging process. If f_i has no term left, then this means that neither term in f_i can be chosen to form a multilinear monomial, so the answer is *"NO"*.

(2) We find that every clause f_j contributes one term π_j during the purging process. This means that $\pi = \pi_1 \cdots \pi_m$ has no variables appearing more than once, hence it is a multilinear monomial, so an answer *"YES"* is obtained.

(3) We find that the purging process fails to purge any terms in a subset of clauses. Let $S \subset I$ denote the set of the indexes of these clauses, where $I = \{1, \ldots, m\}$. Let π' be the product of π_j with $j \in I - S$. According to the purging process, π' does not share any common variables with terms in any clause f_u with $u \in S$. Hence, the input polynomial f has a multilinear monomial iff the product of those clauses f_u has a multilinear monomial. Therefore, we recursively apply the purging process to this product of clauses. Note that this product has at least one fewer clause than f.

With the help of some simple data structure, the purging process can be implemented in quadratic time.

6 $\Pi\Sigma_2\Pi \times \Pi\Sigma$ Polynomials vs. $\Pi\Sigma_2\Pi$ and $\Pi\Sigma$ Polynomials

In structure, a $\Pi\Sigma_2\Pi \times \Pi\Sigma$ polynomial is a product of one $\Pi\Sigma_2\Pi$ polynomial and another $\Pi\Sigma$ polynomial. It has been shown in Sections 4 and 5 that testing multilinear monomials in $\Pi\Sigma_2\Pi$ or $\Pi\Sigma$ polynomials can be done respectively in polynomial time. This might encourage one to think that testing multilinear monomials in $\Pi\Sigma_2\Pi \times \Pi\Sigma$ polynomials could also be done in polynomial time. However, a little bit surprisingly the following theorem shows that a complexity boundary exists, separating $\Pi\Sigma_2\Pi \times \Pi\Sigma$ polynomials from $\Pi\Sigma_2\Pi$ and $\Pi\Sigma$ polynomials.

Theorem 5. *The problem of testing multilinear monomials in $\Pi\Sigma_2\Pi \times \Pi\Sigma$ polynomials is NP-complete.*

Proof. It is easy to see that the given problem is in NP. To show that the problem is also NP-hard, we consider any given $\Pi_m\Sigma_3\Pi_t$ polynomial $f = f_1 \cdots f_m$ with $m \geq 1$ and $t \geq 2$ such that each clause $f_i = (T_{i1}+T_{i2}+T_{i3})$ and each term T_{ij} has degree at most t, $1 \leq i \leq m$, $1 \leq j \leq 3$. We shall reduce f into a $\Pi\Sigma_2\Pi \times \Pi\Sigma$ polynomial. Once this is done, the NP-hardness of the given problem follows from Theorem 1.

We consider the clause

$$f_i = (T_{i1} + T_{i2} + T_{i3}).$$

We want to represent f_i by a $\Pi\Sigma_2\Pi \times \Pi\Sigma$ polynomial so that selecting exactly one term from f_i is equivalent to selecting exactly one monomial from the new polynomial with exactly one term T_{ij} in f_i under the constraint that the newly introduced variables are linear in the monomial. We construct the new polynomial, denoted by $p(f_i)$, as follows.

$$p(f_i) = (T_{i1}u_i + v_i)(T_{i2}u_i + w_i)(T_{i3}u_i + z_i)(v_i + w_i + z_i),$$

where u_i, v_i, w_i and z_i are new variables. It is easy to see that there are only three monomials in $p(f_i)$ satisfying the constraint:

$$T_{i1}u_iv_iw_iz_i, \quad T_{i2}u_iv_iw_iz_i, \quad \text{and} \quad T_{i3}u_iv_iw_iz_i.$$

Each of those three monomials corresponds to exactly one term in f_i. Now, let

$$p(f) = p(f_1) \cdots p(f_m)$$

be the new polynomial representing f and

$$\pi = \pi_1 \cdots \pi_m$$

be a monomial in f with terms π_i in f_i. If π is multilinear, then so is

$$\pi' = (\pi_1 u_1 v_1 w_1 z_1) \cdots (\pi_m u_m v_m w_m z_m)$$

in $p(f)$. On the other hand, if

$$\psi = \psi_1 \cdots \psi_m$$

is multilinear monomial in $p(f)$, then $\psi_i = T_{ij_i}u_iv_iw_iz_i$ with $j_i \in \{1, 2, 3\}$. This implies that

$$\psi' = T_{1j_1} \cdots T_{mj_m}$$

must be a multilinear monomial in f. Obviously, the reduction from f to $p(f)$ can be done in polynomial time.

7 Testing c-Monomials

By definition, a multilinear monomial is a 2-monomial. It has been shown in Section 5 that the problem of testing multilinear monomials in a $\Pi\Sigma_2\Pi$ polynomial is solvable in quadratic time. We shall show that another complexity boundary exists to separate c-monomials from 2-monomials, even when $c = 3$. On the positive side, we shall show that it is efficient to testing c-monomials for $\Pi\Sigma$ polynomials.

Theorem 6. *The problem of testing* 3*-monomials in any* 3-$\Pi_m\Sigma_2\Pi_6$ *polynomial is NP-complete.*

Proof. We only need to show that the problem is NP-hard, since it is trivial to see that the problem is in NP.

Let $f = f_1 \cdots f_m$ be any given 2-$\Pi_m\Sigma_3\Pi_2$ polynomial, where each clause $f_i = (T_{i1} + T_{i2} + T_{i3})$ and each term T_{ij} is multilinear with at most 2 distinct variables, $1 \le i \le m$, $1 \le j \le 3$. By Theorem 1, testing whether f has a multilinear monomial is NP-hard. We now show how to construct a 3-$\Pi_m\Sigma_2\Pi_6$ polynomial to represent f with the property that f has a multilinear monomial iff the new polynomial has a 3-monomial.

We consider the clause

$$f_i = (T_{i1} + T_{i2} + T_{i3}).$$

We want to represent f_i by a 3-$\Pi\Sigma_2\Pi_6$ polynomial so that selecting exactly one term from f_i is equivalent to selecting exactly one 3-monomial from the new polynomial with exactly one term T_{ij} in f_i under the constraints that T_{ij} appears twice and the newly introduced variables are each of degree 2. The idea for constructing the new polynomial seems like what is used in the proof of Theorem 5, but it is different from that construction. We design the new polynomial, denoted by $p(f_i)$, as follows.

$$p(f_i) = (T_{i1}T_{i1}u_i^2 + v_i)(T_{i2}T_{i2}u_i^2 + v_i)(T_{i3}T_{i3}u_i^2 + v_i)$$

where u_i and v_i are new variables. Since each term T_{ij} is multilinear with at most two distinct variables, $p(f_i)$ is a 3-$\Pi_m\Sigma_2\Pi_6$ polynomial. It is easy to see that there are no multilinear monomials in $p(f_i)$. But there are three monomials in $p(f_i)$ satisfying the given constraints:

$$T_{i1}T_{i1}u_i^2v_i^2, \quad T_{i2}T_{i2}u_i^2v_i^2, \quad \text{and} \quad T_{i3}T_{i3}u_i^2v_i^2.$$

Each of those three monomials corresponds to exactly one term in f_i. Note that only those three monomials in $p(f_i)$ can possibly be 3-monomials, depending on whether $T_{ij}T_{ij}$ is a 3-monomials. Now, let

$$p(f) = p(f_i) \cdots p(f_m)$$

be the new polynomial representing f and

$$\pi = \pi_1 \cdots \pi_m$$

be a monomial in f with terms π_i in f_i. If π is multilinear, then

$$\pi' = (\pi_1\pi_1 u_1^2 v_1^2)\cdots(\pi_m\pi_m u_m^2 v_m^2)$$

is a 3-monomial in $p(f)$. On the other hand, if

$$\psi = \psi_1 \cdots \psi_m$$

is a 3-monomial in $p(f)$, then $\psi_i = T_{ij_i}T_{ij_i}u_i^2v_i^2$ with $j_i \in \{1,2,3\}$. This implies that

$$\psi' = T_{1j_1}T_{1j_1}\cdots T_{mj_m}T_{mj_m}$$

is a 3-monomial. Therefore,

$$\psi'' = T_{1j_1}\cdots T_{mj_m}$$

must be a multilinear monomial in f. Obviously, reducing f to $p(f)$ can be done in polynomial time.

The following corollaries follows immediately from Theorem 7:

Corollary 3. *For any $c > 2$, testing c-monomials in any $\Pi_m\Sigma_s\Pi_t$ polynomial is NP-complete.*

Corollary 4. *For any $c > 2$, testing c-monomials in any polynomial represented by a formula or a general arithmetic circuit is NP-complete.*

Recall that by Theorem 2 the multilinear monomial testing problem for $\Pi\Sigma$ polynomials is solvable in polynomial time. The following theorem shows a complementary result about c-monomial testing for the same type of polynomials.

Theorem 7. *There is a $O(cms\sqrt{m+cn})$ time algorithm to test whether any $\Pi_m\Sigma_s$ polynomial has a c-monomial or not, where $c > 2$ is a fixed constant.*

Proof. We consider to generalize the maximum matching reduction in Theorem 2. Like before, Let $f(x_1,\ldots,x_n) = f_1\ldots f_m$ be any given $\Pi_m\Sigma_s$ polynomial such that $f_i = \sum_{j=1}^{s} x_{i_j}$, $1 \le i \le s$. We construct a bipartite graph $G = (V_1 \cup V_2, E)$ as follows. $V_1 = \{v_1,\ldots,v_m\}$ so that each v_i represents the clause f_i. $V_2 = \cup_{i=1}^{n}\{u_{i1},u_{i2},\ldots,u_{i(c-1)}\}$, i.e., each variable x_i corresponds to $c-1$ vertices $u_{i1},u_{i2},\ldots,u_{i(c-1)}$. For each clause f_i, if it contains a variable x_j then we add $c-1$ edges (v_i,u_{jt}) into E, $1 \le t \le c-1$.

Suppose that $f(x_1,\ldots,x_n)$ has a c-monomial

$$\pi = x_{i_1}\cdots x_{i_m}$$

with x_{i_j} in f_j, $1 \le j \le m$. Note that each variable x_{i_j} appears $k(x_{i_j}) < c$ times in π. Those appearances correspond to $k(x_{i_j})$ clauses $f_{t_1},\ldots,f_{t_{k(x_{i_j})}}$ from which x_{i_j} was respectively selected to form π. This implies that there are $k(x_{i_j})$ edges matching $v_{t_1},\ldots,v_{t_{k(x_{i_j})}}$ with $k(x_{i_j})$ vertices in V_2 that represent x_{i_j}. Hence, the collection of m edges for m appearances of all the variables, repeated or not, in π forms a maximum matching of size m in the graph G.

Now, assume that we have a maximum matching of size m

$$(v_1, u_{i_1 j_1}), \ldots, (v_m, u_{i_m j_m}).$$

Recall that $u_{i_t j_t}$, $1 \le t \le m$, is designed to represent the variable x_{i_t}. By the construction of the graph G, x_{i_t} are in the clause f_t, $1 \le t \le m$, and it may appear $c-1$ times. Hence,

$$\pi = x_{i_1} \cdots x_{i_m}$$

is a c-monomial in $f(x_1, \ldots, x_n)$.

With the help of the $O(|E|\sqrt{|V|})$ time algorithm [10] for finding a maximum matching in a bipartite graph, testing whether $f(x_1, \ldots, x_n)$ has a c-monomial can done in $O(cms\sqrt{m+cn})$, since the graph G has $m+cn$ vertices and at most cms edges.

8 Parameterized Algorithms

In this section, we shall devise two parameterized algorithms for testing multilinear monomials in $\Pi_m\Sigma_3\Pi_t$ and $\Pi_m\Sigma_2\Pi_t \times \Pi_k\Sigma_3$ polynomials. By Theorems 1 and 5, the multilinear monomial testing problem for each of these two types of polynomials is NP-complete.

Theorem 8. *There is a $O(tm^2 1.7751^m)$ time algorithm to test whether any $\Pi_m\Sigma_3\Pi_t$ polynomial has a multilinear monomial in its sum-product expansion.*

Proof. Let $f = f_1 \cdots f_m$ be any given $\Pi_m\Sigma_3\Pi_t$ polynomial, where each clause

$$f_i = (T_{i1} + T_{i2} + T_{i3})$$

and each term T_{ij} has degree at most t, $1 \le i \le m$, $1 \le j \le 3$.

We now consider to reduce f to an undirected graph $G = (V, E)$ such that f has a multilinear monomial iff G has a maximum m-clique. For each clause f_i, we design three vertices v_{i1}, v_{i2} and v_{i3}, representing the three corresponding terms in f_i. Let V be the collection of those vertices for all the terms in f. For any two vertices v_{ij} and $v_{i'j'}$ with $i \ne i'$, we add an edge $(v_{ij}, v_{i'j'})$ to E, if their corresponding terms T_{ij} and $T_{i'j'}$ do not share any common variable. Since any two vertices designed for the terms in a clause are not connected, the maximum cliques in G could have m vertices corresponding to m terms, each of which is in one of those m clauses. Let

$$\pi = \pi_1 \cdots \pi_m$$

be any monomial in f with π being a term from f_i. We consider two cases in the following.

Assume that π is a multilinear monomial. Let $\pi_i = T_{ij_i}$, $j_i \in \{1,2,3\}$. Then, any two terms T_{ij_i} and $T_{i'j_{i'}}$ in π do not share any common variable. So, there is an edge $(v_{ij_i}, v_{i'j_{i'}})$ in E. Hence, the graph G has an m-clique $\{v_{1j_1}, \ldots, v_{mj_m}\}$. Certainly, this clique is maximum.

Now, suppose that G has a maximum clique $\{v_{1j_1}, \ldots, v_{mj_m}\}$. Then, by the construction of G, each vertex v_{ij_i} corresponds to the term T_{ij_i} in the clause f_i. Thus, the product of those m terms is a multilinear monomial, because any two of those terms do not share a common variable.

Finally, we use Robson's $O(1.2108^{|V|})$ algorithm to find a maximum clique for G. If the clique has size m, then f has a multilinear monomial. Otherwise, it does not. Note that $|V| = 3m$. Combining the reduction time with the clique finding time gives an overall $O(tm^2 1.7751^m)$ time.

We now turn to $\Pi_m\Sigma_2\Pi_t \times \Pi_k\Sigma_3$ polynomials and give the second parameterized algorithm for this type of polynomials.

Theorem 9. *There is a $O((mk)^2 3^k)$ time algorithm to test whether any $\Pi_m\Sigma_2\Pi_t \times \Pi_k\Sigma_3$ polynomial has a multilinear monomial in its sum-product expansion.*

Proof. Let $p = p_1 \cdot p_2$ such that p_1 is a $\Pi_m\Sigma_2\Pi_t$ polynomial and p_2 is a $\Pi_k\Sigma_3$ polynomial. In $O(3^k)$ time, we list all the products in the sum-product expansion of p_2. Let $\mathcal{C}$ be the collection of those products. It is obvious that p has a multilinear monomial iff there is a product $\pi \in \mathcal{C}$ such that $p_1 \cdot \pi$ has a multilinear monomial. Note that $p_1 \cdot \pi$ is a $\Pi_{m+1}\Sigma_2\Pi_t$ polynomial. By Theorem 4, the multilinear monomial testing problem for $p_1 \cdot \pi$ can be solved by a quadratic time algorithm. Hence, the theorem follows by applying that algorithm to $p_1 \cdot \pi$ for every $\pi \in \mathcal{C}$ to see if one of them has a multilinear monomial or not.

Acknowledgments. We thank Yang Liu and Robbie Schweller for many valuable discussions during our weekly seminar. Conversations with them help inspire us to develop this study of testing monomials. We thank Yang Liu for presenting Koutis' paper [13] at the seminar. The $O((ms)^2 3^k)$ upper bound given in Theorem 9 has been improved to $O((ms)^2 2^k)$ in [7], along with several randomized and deterministic algorithms. The complexity of approximating monomial coefficients are studies in citechen11a.

Bin Fu's research is support by an NSF CAREER Award, 2009 April 1 to 2014 March 31.

References

1. Manindra, A., Neeraj, K., Nitin, S.: PRIMES is in P. Ann. of Math. 160(2), 781–793 (2004)
2. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. Journal of the ACM 45(3), 501–555 (1998)
3. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters 8(3), 121–123 (1979)

4. Beigel, R.: The polynomial method in circuit complexity. In: Proceedings of the Eighth Conference on Structure in Complexity Theory, pp. 82–95 (1993)
5. Bshouty, N.H., Chen, Z., Decatur, S.E., Homer, S.: One the learnability of Z_N-DNF formulas. In: Proceedings of the Eighth Annual Conference on Computational Learning Theory (COLT 1995), pp. 198–205. ACM, Santa Cruz (1995)
6. Chen, Z., Fu, B.: Approximating multilinear monomial coefficients and maximum multilinear monomials in multivariate polynomials. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part I. LNCS, vol. 6508, pp. 309–323. Springer, Heidelberg (2010)
7. Chen, Z., Fu, B., Liu, Y., Schweller, R.T.: Algorithms for Testing Monomials in Multivariate Polynomials. In: Proceedings of the Fifth International Conference on Combinatorial Optimization and Applications (2011)
8. Feige, U., Goldwasser, S., Lovász, L., Safra, S., Szegedy, M.: Interactive proofs and the hardness of approximating cliques. Journal of the ACM 43(2), 268–292 (1996)
9. Fu, B.: Separating PH from PP by relativization. Acta Math. Sinica 8(3), 329–336 (1992)
10. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM Journal on Computing 2(4), 225–231 (1973)
11. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. In: STOC, pp. 355–364 (2003)
12. Klivans, A., Servedio, R.A.: Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. In: STOC, pp. 258–265 (2001)
13. Koutis, I.: Faster algebraic algorithms for path and packing problems. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 575–586. Springer, Heidelberg (2008)
14. Minsky, M., Papert, S.: Perceptrons (expanded edition 1988). MIT Press, Cambridge (1968)
15. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Reading (1994)
16. Robson, J.M.: Algorithms for maximum independent sets. Journal of Algorithms 7(3), 425–440 (1986)
17. Shamir, A.: IP = PSPACE. Journal of the ACM 39(4), 869–877 (1992)
18. Williams, R.: Finding paths of length k in $O^*(2^k)$ time. Information Processing Letters 109, 315–318 (2009)

Algorithms for Testing Monomials in Multivariate Polynomials

Zhixiang Chen, Bin Fu, Yang Liu, and Robert Schweller

Department of Computer Science,
University of Texas-Pan American,
Edinburg, TX 78539, USA
{chen,binfu,yliu,schwellerr}@cs.panam.edu

Abstract. This paper is our second step towards developing a theory of testing monomials in multivariate polynomials. The central question is to ask whether a polynomial represented by an arithmetic circuit has some types of monomials in its sum-product expansion. The complexity aspects of this problem and its variants have been investigated in our first paper by Chen and Fu (2010), laying a foundation for further study. In this paper, we present two pairs of algorithms. First, we prove that there is a randomized $O^*(p^k)$ time algorithm for testing p-monomials in an n-variate polynomial of degree k represented by an arithmetic circuit, while a deterministic $O^*((6.4p)^k)$ time algorithm is devised when the circuit is a formula, here p is a given prime number. Second, we present a deterministic $O^*(2^k)$ time algorithm for testing multilinear monomials in $\Pi_m\Sigma_2\Pi_t \times \Pi_k\Sigma_3$ polynomials, while a randomized $O^*(1.5^k)$ algorithm is given for these polynomials. Finally, we prove that testing some special types of multilinear monomial is W[1]-hard, giving evidence that testing for specific monomials is not fixed-parameter tractable.

Keywords: Multivariate polynomials, monomial testing, algebra, complexity, randomization, derandomization.

1 Introduction

1.1 Overview

We begin with the k-path problem to exhibit the motivation and necessity of the study about the monomial testing problem for multivariate polynomials. Throughout this paper, polynomials refer to those with multiple variables. Let $G = (V, E)$ be an undirected graph with $|V| = n$. For any fixed integer $c \geq 1$, for each vertex $v_i \in V$, define a polynomial $F_{k,i}$ as follows:

$$F_{1,i} = x_i^c,$$
$$F_{k+1,i} = x_i^c \left(\sum_{(v_i,v_j)\in E} F_{k,j} \right), \ k > 1.$$

We define a polynomial for G as

$$F(G,k) = \sum_{i=1}^{n} F_{k,i}.$$

Obviously, $F(G,k)$ can be represented by an arithmetic circuit. It is easy to see that the graph G has a k-path $v_{i_1} \cdots v_{i_k}$ iff $F(G,k)$ has a monomial $x_{i_1}^c \cdots x_{i_k}^c$ of degree ck in its sum-product expansion. G has a Hamiltonian path iff $F(G,n)$ has the monomial $x_1^c \cdots x_n^c$ of degree cn in its sum-product expansion. One can also see that a path with some loop can be characterized by a monomial as well. Those observations show that testing monomials in polynomials is closely related to solving k-path, Hamiltonian path and other problems about graphs. When $c = 1$, $x_{i_1} \cdots x_{i_k}$ is multilinear. The problem of testing multilinear monomials has recently been exploited by Koutis [14] and Williams [20] to design innovative randomized parameterized algorithms for the k-path problem.

There has been a long history in theoretical computer science with heavy involvement of studies and applications of polynomials. Most notably, low degree polynomial testing/representing and polynomial identity testing have played invaluable roles in many major breakthroughs in complexity theory. For example, low degree polynomial testing is involved in the proof of the PCP Theorem, the cornerstone of the theory of computational hardness of approximation and the culmination of a long line of research on IP and PCP (see, Arora *et al.* [3] and Feige *et al.* [10]). Polynomial identity testing has been extensively studied due to its role in various aspects of theoretical computer science (see, for example, Kabanets and Impagliazzo [12]) and its applications in various fundamental results such as Shamir's IP=PSPACE [19] and the AKS Primality Testing [2]. Low degree polynomial representing [15] has been sought for so as to prove important results in circuit complexity, complexity class separation and subexponential time learning of boolean functions (see, for examples, Beigel [5], Fu[11], and Klivans and Servedio [13]). These are just a few examples. A survey of the related literature is certainly beyond the scope of this paper.

The above example of the k-path testing, the rich literature about polynomial testing and many other examples and observations have motivated us to develop a new theory of testing monomials in polynomials represented by arithmetic circuits or even simpler structures. The monomial testing problem is related to, and somehow complements with, the low degree testing and the identity testing of polynomials. We want to investigate various complexity aspects of the monomial testing problem and its variants with two folds of objectives. One is to understand how this problem relates to critical problems in complexity, and if so to what extent. The other is to exploit possibilities of applying algebraic properties of polynomials to the study of those critical problems. As a first step, Chen and Fu [6] have proved a series of results: The multilinear monomial testing problem for $\Pi\Sigma\Pi$ polynomials is NP-hard, even when each clause has at most three terms. The testing problem for $\Pi\Sigma$ polynomials is in P, and so is the testing for two-term $\Pi\Sigma\Pi$ polynomials. However, the testing for a product of one two-term $\Pi\Sigma\Pi$ polynomial and another $\Pi\Sigma$ polynomial is NP-hard. This type of polynomial products is, more or less, related to the polynomial factorization problem. We have also proved that testing c-monomials for two-term $\Pi\Sigma\Pi$ polynomials is NP-hard for any $c > 2$, but the same testing is in P for $\Pi\Sigma$ polynomials. Finally, two parameterized algorithms have been devised for three-term $\Pi\Sigma\Pi$ polynomials and products of two-term $\Pi\Sigma\Pi$ and $\Pi\Sigma$ polynomials. These results have laid a basis for further study about testing monomials.

1.2 Contributions and Methods

The major contributions of this paper are two pairs of algorithms. For the first pair, we prove that there is a randomized $O^*(p^k)$ time algorithm for testing p-monomials in an n-variate polynomial of degree k represented by an arithmetic circuit, while a deterministic $O^*((6.4p)^k)$ time algorithm is devised when the circuit is a formula, here p is a given prime number. The first algorithm extends two recent algorithms for testing multilinear monomials, the $O^*(2^{3k/2})$ algorithm by Koutis [14] and the $O(2^k)$ algorithm by Williams [20]. Koutis [14] initiated the application of group algebra $Z_2[Z_2^k]$ to randomized testing of multilinear monomials in a polynomial. Williams [20] incorporated the randomized Schwartz-Zippel polynomial identity testing with the group algebra $\mathrm{GF}(2^\ell)[Z_p^k]$ for some relatively small ℓ in comparison with k to achieve the design of his algorithm. The success of applying group algebra to designing multilinear monomial testing algorithms is based on two simple but elegant properties found by Koutis, by which annihilating non-multilinear monomials is possible via replacements of variables by vectors in Z_2^k. When extending the group algebra from $Z_2[Z_p^k]$ to $Z_p[Z_p^k]$ for a given prime p these two properties, as addressed in Section 3, are unfortunately no longer valid. To make the matter worse, the Schwartz-Zippel algorithm is not applicable to the larger algebra due to the lack of these two properties. Nevertheless, we find new characteristics about $Z_p[Z_p^k]$ and integrate these with a more powerful randomized polynomial identity testing algorithm by Agrawal and Biswas [1] to accomplish the design of our algorithm. Our deterministic algorithm is obtained via derandomizing the two random processes involved in the first algorithm: deterministic selection of a set of linearly independent vectors for an unknown monomial to guarantee its survivability from vector replacements; and deterministic polynomial identity testing. The first part is realized with the perfect hashing functions by Chen *et al.* [8], while the second is carried out by the Raz and Shpilka [18] algorithm for noncommunicative polynomials.

For the second pair of our algorithms, we present a deterministic $O^*(2^k)$ time algorithm for testing multilinear monomials in $\Pi_m\Sigma_2\Pi_t \times \Pi_k\Pi_3$ polynomials, while a randomized $O^*(1.5^k)$ algorithm is given for these polynomials. It has been proved in Chen and Fu [6] that testing multilinear monomials in $\Pi_m\Sigma_2\Pi_t$ or $\Pi_k\Pi_3$ polynomials is solvable in polynomial time. However, the problem becomes NP-hard for $\Pi_m\Sigma_2\Pi_t \times \Pi_k\Sigma_3$ polynomials. Our two algorithms use the quadratic algorithm by Chen and Fu [6] for testing multilinear monomials in $\Pi_m\Sigma_2\Pi_t$ polynomials as the base case algorithm. Both new algorithms improve the $O^*(3^k)$ algorithm in [6].

Finally, we prove that testing some special types of multilinear monomials is W[1]-hard, giving evidence that testing for specific monomials is not fixed-parameter tractable. One shall notice that difference between the general monomial testing and the specific monomial testing. The former asks for the existence of *"any one"* from a set of possibly many monomials that are needed. The latter asks for *"a specific one"* from the set.

2 Preliminaries

2.1 Notations and Definitions

For variables $x_1, \ldots, x_n$, let $\mathcal{P}[x_1, \cdots, x_n]$ denote the communicative ring of all the n-variate polynomials with coefficients from a finite field $\mathcal{P}$. For $1 \leq i_1 < \cdots < i_k \leq n$, $\pi = x_{i_1}^{j_1} \cdots x_{i_k}^{j_k}$ is called a monomial. The degree of π, denoted by $\deg(\pi)$, is $\sum_{s=1}^{k} j_s$. π is multilinear, if $j_1 = \cdots = j_k = 1$, i.e., π is linear in all its variables $x_{i_1}, \ldots, x_{j_k}$. For any given integer $c \geq 2$, π is called a c-monomial, if $1 \leq j_1, \ldots, j_k < c$.

An arithmetic circuit, or circuit for short, is a direct acyclic graph with $+$ gates of unbounded fan-in, $\times$ gates of fan-in two, and all terminals corresponding to variables. The size, denoted by $s(n)$, of a circuit with n variables is the number of gates in it. A circuit is called a formula, if the fan-out of every gate is at most one, i.e., its underlying direct acyclic graph is a tree.

Throughout this paper, the $O^*(\cdot)$ notation is used to suppress poly(n, k) factors in time complexity bounds.

Definition 1. *Let $F(x_1, \ldots, x_n) \in \mathcal{P}[x_1, \ldots, x_n]$ be any given polynomial. Let $m, s, t \geq 1$ be integers.*

- *$F(x_1, \ldots, x_n)$ is said to be a $\Pi_m \Sigma_s \Pi_t$ polynomial, if $F(x_1, \ldots, x_n) = \prod_{i=1}^{m} F_i$, $F_i = \sum_{j=1}^{r_i} X_{ij}$ and $1 \leq r_i \leq s$, and X_{ij} is a product of variables with $deg(X_{ij}) \leq t$. We call each F_i a clause. Note that X_{ij} is not a monomial in the sum-product expansion of $p(x_1, \ldots, x_n)$ unless $m = 1$. To differentiate this subtlety, we call X_{ij} a term.*
- *In particular, we say $F(x_1, \ldots, x_n)$ is a $\Pi_m \Sigma_s$ polynomial, if it is a $\Pi_m \Sigma_s \Pi_1$ polynomial. Here, each clause in f is a linear addition of single variables. In other word, each term has degree 1.*
- *$F(x_1, \ldots, x_n)$ is called a $\Pi_m \Sigma_s \Pi_t \times \Pi_k \Sigma_\ell$ polynomial, if $F(x_1, \ldots, x_n) = f_1 \cdot f_2$ such that f_1 is a $\Pi_m \Sigma_s \Pi_t$ polynomial and f_2 is a $\Pi_k \Sigma_\ell$ polynomial.*

When no confusion arises from the context, we use $\Pi\Sigma\Pi$ and $\Pi\Sigma$ to stand for $\Pi_m \Sigma_s \Pi_t$ and $\Pi_m \Sigma_s$, respectively.

2.2 The Group Algebra $\boldsymbol{F}[\boldsymbol{Z}_p^k]$

For any prime p and integer $k \geq 2$, we consider the group Z_p^k with the multiplication $\cdot$ defined as follows. For k-dimensional column vectors $\boldsymbol{x}, \boldsymbol{y} \in Z_p^k$ with

$\boldsymbol{x} = (x_1, \ldots, x_k)^T$ and $\boldsymbol{y} = (y_1, \ldots, y_k)^T$, $\boldsymbol{x} \cdot \boldsymbol{y} = (x_1 + y_1 \pmod p), \ldots, x_k + y_k \pmod p)^T$. $\mathbf{0} = (0, \ldots, 0)^T$ is the zero element in the group. For any field F, the group algebra $F[Z_p^k]$ is defined as follows. Every element $u \in F[Z_p^k]$ is a linear addition of the form

$$u = \sum_{\boldsymbol{x} \in Z_p^k, a_{\boldsymbol{x}} \in F} a_{\boldsymbol{x}} \boldsymbol{x}. \tag{1}$$

For any element $v = \sum_{\boldsymbol{x} \in Z_p^k, b_{\boldsymbol{x}} \in F} b_{\boldsymbol{x}} \boldsymbol{x}$, We define

$$u + v = \sum_{a_{\boldsymbol{x}}, b_{\boldsymbol{x}} \in F,\ \boldsymbol{x} \in Z_p^k} (a_{\boldsymbol{x}} + b_{\boldsymbol{x}} \pmod p)\boldsymbol{x}, \text{ and}$$
$$u \cdot v = \sum_{a_{\boldsymbol{x}}, b_{\boldsymbol{y}} \in F, \text{ and } \boldsymbol{x}, \boldsymbol{y} \in Z_p^k} (a_{\boldsymbol{x}} b_{\boldsymbol{y}} \pmod p)(\boldsymbol{x} \cdot \boldsymbol{y}).$$

For any scalar $w \in F$,

$$wu = w \left(\sum_{\boldsymbol{x} \in Z_p^k,\ a_{\boldsymbol{x}} \in F} a_{\boldsymbol{x}} \boldsymbol{x} \right) = \sum_{\boldsymbol{x} \in Z_p^k,\ a_{\boldsymbol{x}} \in F} (w a_{\boldsymbol{x}} \pmod p)\boldsymbol{x}.$$

The zero element in $F[Z_p^k]$ is the one as represented in expression (1) with zero coefficients in F: $\mathbf{0} = \sum_{\boldsymbol{x} \in Z_p^k} 0\boldsymbol{x} = 0\mathbf{0}$. The identity element in $F[Z_p^k]$ is $\mathbf{1} = 1\mathbf{0} = \mathbf{0}$. For any vector $\boldsymbol{v} = (v_1, \ldots, v_k)^T \in Z_p^k$, for $i \geq 0$, let $(\boldsymbol{v})^i = (iv_1 \pmod p), \ldots, iv_k \pmod p)^T$. In particular, we have $(\boldsymbol{v})^0 = (\boldsymbol{v})^p = \mathbf{0}$. When it is clear from the context, we will simply use xy and $x + y$ to stand for $xy (\bmod p)$ and $x + y \pmod p$, respectively.

3 Randomized Testing of p-Monomials

Group algebra $Z_2[Z_2^k]$ was first used by Koutis [14] and later by Williams [20] to devise a randomized $O^*(2^k)$ time algorithm to test multilinear monomials in n-variate polynomials represented by arithmetic circuits. We shall extend $Z_2[Z_2^k]$ to $Z_p[Z_p^d]$ to test p-monomials for some $d > k$. Two key properties in $Z_2[Z_2^k]$, as first found by Koutis [14], that are crucial to multilinear monomial testing are unfortunately no longer valid in $Z_p[Z_p^d]$. Instead, we establish new properties in Lemmas 3 and 4. Also, the Schwartz-Zippel algorithm [16] for randomized polynomial identity testing adopted by Williams [20] is not applicable to our case. Instead, we have to use a more advanced randomized polynomial identity testing algorithm, the Agrawal and Biswas algorithm [1].

Let p be a prime number. Following conventional notations in linear algebra, for any vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t \in Z_p^k$ with $k \geq 1$ and $t \geq 1$, let $\text{span}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t)$ be the linear space spanned by these vectors. That is, $\text{span}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t) = \{a_1\boldsymbol{v}_1 + \cdots + a_t\boldsymbol{v}_t | a_1, \ldots, a_t \in Z_p\}$.

We first give two simple properties about $(\text{mod} p)$ operation.

Lemma 1. *For any $x, y \in Z_p$, we have $(x+y)^p = x^p + y^p \pmod{p}$.*

Proof. $(x+y)^p = \sum_{i=0}^{p} \binom{p}{i} x^{p-i} y^i = x^p + y^p + \sum_{i=1}^{p-1} \binom{p}{i} x^{p-i} y^i$. Since p is prime, $\binom{p}{i}$ has a factor p, implying $\binom{p}{i} = 0 \pmod{p}$, $1 \le i \le p-1$. Hence, $(x+y)^p = x^p + y^p \pmod{p}$.

Lemma 2. *For any $x, y \in Z_p$, we have $((p-1)x+y)^p = (p-1)x^p + y^p \pmod{p}$.*

Proof. By Lemma 1, $((p-1)x+y)^p \equiv (p-1)^p x^p + y^p \pmod{p}$. By Fermat's Little Theorem, $(p-1)^p = (p-1) \pmod{p}$. Thus, $((p-1)x+y)^p = (p-1)x^p + y^p \pmod{p}$.

The first crucial, though simple, property observed by Koustis [14] about testing multilinear monomials is that replacing any variable x by $(\boldsymbol{v} + \boldsymbol{0})$ will annihilate x^t for any $t \ge 2$, where $\boldsymbol{v} \in Z_2^k$ and $\boldsymbol{v}_0$ is the zero vector. This property is not valid in $Z_p[Z_p^d]$. However, we shall prove the following lemma that helps annihilate any monomials that are not p-monomials.

Lemma 3. *Let $\boldsymbol{v}_0 \in Z_p^d$ be the zero vector and $\boldsymbol{v}_i \in Z_p^d$ be any vector. Then, we have*

$$((p-1)\boldsymbol{v}_i + \boldsymbol{v}_0)^p = \boldsymbol{0}, \tag{2}$$

i.e., the zero element in $Z_p[Z_p^d]$.

Proof. By Lemma 2, we have $((p-1)\boldsymbol{v}_i + \boldsymbol{v}_0)^p = (p-1)(\boldsymbol{v}_i)^p + (\boldsymbol{v}_0)^p \pmod{p} = (p-1)\boldsymbol{v}_0 + \boldsymbol{v}_0 = p\boldsymbol{v}_0 \pmod{p} = \boldsymbol{0}$.

The second crucial property found by Koutis [14] has two parts: (a) Replacing variables x_{i_j} in a multilinear monomial $x_{i_1} \cdots x_{i_k}$ with $(\boldsymbol{v}_{i_j} + \boldsymbol{v}_0)$ will annihilate the monomial, if the vectors $\boldsymbol{v}_{i_j}$ are linearly dependent in Z_2^k. (b) If these vectors are linearly independent, then the sum-product expansion of the monomial after the replacements will yield a sum of all 2^k vectors in Z_2^k. However, neither (a) nor (b) is in general true in $Z_p[Z_p^k]$. Fortunately, we have the following lemma, though not as *"structurally"* perfect as (b).

Lemma 4. *Let $x_1^{m_1} \cdots x_t^{m_t}$ be any given p-monomial of degree k. If vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t \in Z_p^d$ are linearly independent, then there are nonzero coefficients $c_i \in Z_p$ and pairwise distinct vectors $\boldsymbol{u}_i \in Z_p^d$ such that*

$$((p-1)\boldsymbol{v}_1 + \boldsymbol{v}_0)^{m_1} \cdots ((p-1)\boldsymbol{v}_t + \boldsymbol{v}_0)^{m_t} = \sum_{i=1}^{(m_1+1)(m_2+1)\cdots(m_t+1)} c_i \boldsymbol{u}_i, \tag{3}$$

where $c_1 = 1$ and $\boldsymbol{u}_1 = \boldsymbol{0}$.

Proof.

$$\begin{aligned}
&((p-1)\boldsymbol{v}_1 + \boldsymbol{v}_0)^{m_1} \cdots ((p-1)\boldsymbol{v}_t + \boldsymbol{v}_0)^{m_t} \\
&= \left(\sum_{i_1=0}^{m_1} \binom{m_1}{i_1} (p-1)^{i_1} (\boldsymbol{v}_1)^{i_1} \right) \left(\sum_{i_2=0}^{m_2} \binom{m_2}{i_2} (p-1)^{i_2} (\boldsymbol{v}_2)^{i_2} \right) \cdots \left(\sum_{i_t=0}^{m_t} \binom{m_t}{i_t} (p-1)^{i_t} (\boldsymbol{v}_t)^{i_t} \right) \\
&= \sum_{i_1=0}^{m_1} \sum_{i_2=0}^{m_2} \cdots \sum_{i_t=0}^{m_t} \binom{m_1}{i_1}\binom{m_2}{i_2} \cdots \binom{m_t}{i_t} (p-1)^{i_1+t_2\cdots+i_t} (\boldsymbol{v}_1)^{i_1} (\boldsymbol{v}_2)^{i_2} \cdots (\boldsymbol{v}_t)^{i_t}
\end{aligned} \tag{4}$$

As noted in the previous section, in the vector space Z_p^d, we have

$$(\boldsymbol{v}_1)^{i_1}(\boldsymbol{v}_2)^{i_2}\cdots(\boldsymbol{v}_t)^{i_t} = i_1\boldsymbol{v}_i + i_2\boldsymbol{v}_2 + \cdots + t_t\boldsymbol{v}_t. \tag{5}$$

Since $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_t$ are linearly independent, by expression (5) we have

$$(\boldsymbol{v}_1)^{i_1}(\boldsymbol{v}_2)^{i_2}\cdots(\boldsymbol{v}_t)^{i_t} = \mathbf{0} \text{ iff } i_1 = i_2 = \cdots = i_t = 0. \tag{6}$$

The linear independence of $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_t$ implies that any non-empty subset of these vectors are also linearly independent. Similar to expression (6), this further implies that, for any $0 \le j_i \le m_i$, $i = 1, 2, \ldots, t$,

$$\begin{aligned}&(\boldsymbol{v}_1)^{i_1}(\boldsymbol{v}_2)^{i_2}\cdots(\boldsymbol{v}_t)^{i_t} = (\boldsymbol{v}_1)^{j_1}(\boldsymbol{v}_2)^{j_2}\cdots(\boldsymbol{v}_t)^{j_t}\\ &\text{iff } i_1 = j_1, i_2 = j_2, \ldots, \text{ and } i_t = j_t.\end{aligned} \tag{7}$$

Furthermore, since p is prime and $m_i \in Z_p$, we have

$$\begin{aligned}c(i_1, i_2, \ldots c_t) &= \tbinom{m_1}{i_1}\tbinom{m_2}{i_2}\cdots\tbinom{m_t}{i_t}(p-1)^{i_1+t_2\cdots+i_t} \pmod p\\ &\ne 0 \pmod p\end{aligned} \tag{8}$$

Combining expressions (7) and (8), we have

$$\begin{aligned}&((p-1)\boldsymbol{v}_1 + \boldsymbol{v}_0)^{m_1}\cdots((p-1)\boldsymbol{v}_t + \boldsymbol{v}_0)^{m_t}\\ &= \sum_{0\le i_j\le m_j, 0\le j\le t, i_1+i_2\cdots+i_t\ge 0} c(i_1, i_2, \ldots, i_t)\cdot((\boldsymbol{v}_1)^{i_1}(\boldsymbol{v}_2)^{i_2}\cdots(\boldsymbol{v}_t)^{i_t}).\end{aligned} \tag{9}$$

In the above expression (9), all the coefficients are nonzero, and all the $(m_1 + 1)(m_2+1)\cdots(m_t+1) \le p^k$ vectors are distinct. Hence, expression (3) is obtained.

Theorem 1. *Let p be a prime number. Let $F(x_1, x_2, \ldots, x_n)$ be an n-variate polynomial of degree k represented by an arithmetic circuit C of size $s(n)$. There is a randomized $O^*(p^k)$ time algorithm to test with high probability whether F has a p-monomial of degree k in its sum-product expansion.*

Proof. Let $d = k + \log_p k + 1$, we consider the group algebra $Z_p[Z_p^d]$. As in Williams [20], we first expand the circuit C to a new circuit C' as follows. For each multiplication gate g_i, we attach a new gate g_i' that multiplies the output of g_i with a new variable y_i, and feed the output of g_i' to the gate that reads the output of g_i. Assume that C has h multiplication gates. Then, C' will have h new multiplication gates corresponding to new variables $y_1, y_2, \ldots, y_h$. Let $F'(y_1, y_2, \ldots, y_h, x_1, x_2, \ldots, x_n)$ be he new polynomial represented by C'. The algorithm for testing whether F has a p-monomial of degree k is given in the following.

```
Algorithm RT-MLM (Randomized Testing of Multilinear Monomials):
 1. Select uniform random vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in Z_p^d - \{\mathbf{0}\}$.
 2. Replace each variable $x_i$ with $(\boldsymbol{v}_i + \boldsymbol{v}_0)$, $1 \le i \le n$.
```

3. Use C' to calculate

$$
\begin{aligned}
&F'(y_1,\ldots,y_h,(\boldsymbol{v}_1+\boldsymbol{v}_0),\ldots,(\boldsymbol{v}_n+\boldsymbol{v}_0))\\
&=\sum_{j=1}^{2^d} f_j(y_1,\ldots,y_h)\cdot \boldsymbol{z}_j, \qquad (10)
\end{aligned}
$$

where each f_j is a polynomial of degree k over the finite field Z_p, and $\boldsymbol{z}_j$ with $1\le j\le 2^d$ are the 2^d distinct vectors in Z_p^d.
4. Perform polynomial identity testing with the Agrawal and Biswas algorithm [1] for every f_j over Z_p. Return *"yes"* if one of them is not identical to zero, or *"no"* otherwise.

It follows from Lemma 3 that all monomials that are not p-monomials in F (and hence in F') will become zero, when variables x_i is replaced by $(\boldsymbol{v}_i+\boldsymbol{v}_0)$ at Step ii. We shall estimate that with high probability some p-monomials will survive from those replacements, i.e., will not become the zero element $\mathbf{0}$ in $Z_p[Z_p^d]$.

Consider any given p-monomial $\pi = x_{i_1}^{m_1}\cdots x_{i_t}^{m_t}$ of degree k with $1\le m_i<p$ and $k=m_1+\cdots+m_t$, $i=1,\ldots,t$. For any $1\le j\le t$,

$$
\Pr\left[\boldsymbol{v}_j\in \mathrm{span}(\boldsymbol{v}_{i_1},\ldots,\boldsymbol{v}_{i_{j-1}})\right]=\frac{p^{j-1}}{p^d},
$$

since $|\mathrm{span}(\boldsymbol{v}_{i_1},\ldots,\boldsymbol{v}_{i_{j-1}})|=p^{j-1}$ and $|Z_p^d|=p^d$. Hence,

$$
\begin{aligned}
&\Pr\left[(\exists j\in\{1,\ldots,t\})[\boldsymbol{v}_{i_j}\in \mathrm{span}(\boldsymbol{v}_{i_1},\ldots,\boldsymbol{v}_{i_{j-1}})]\right]\\
&=\Pr\left[[\boldsymbol{v}_1=\mathbf{0}]\vee[\boldsymbol{v}_{i_2}\in\mathrm{span}(\boldsymbol{v}_{i_1})]\vee\cdots\vee[\boldsymbol{v}_{i_t}\in\mathrm{span}(\boldsymbol{v}_{i_1},\ldots,\boldsymbol{v}_{i_{t-1}})]\right]\\
&\le \Pr[\boldsymbol{v}_1=\mathbf{0}]+\Pr[\boldsymbol{v}_{i_2}\in\mathrm{span}(\boldsymbol{v}_{i_1})]+\cdots+\Pr[\boldsymbol{v}_{i_t}\in\mathrm{span}(\boldsymbol{v}_{i_1},\ldots,\boldsymbol{v}_{i_{t-1}})]\\
&=\frac{p^0}{p^d}+\frac{p^1}{p^d}+\cdots+\frac{p^{t-1}}{p^d}\le t\frac{p^{t-1}}{p^d}\\
&\le k\frac{p^{k-1}}{p^{k+\log_p k+1}}\le\frac{1}{p^2}\le\frac{1}{4}. \qquad (11)
\end{aligned}
$$

Because $\boldsymbol{v}_{i_1},\ldots,\boldsymbol{v}_{i_t}$ are linearly independent iff there is no $\boldsymbol{v}_{i_j}\in\mathrm{span}(\boldsymbol{v}_{i_1},\ldots,\boldsymbol{v}_{i_{j-1}})$, by expression (11) the probability that $\boldsymbol{v}_{i_1},\ldots,\boldsymbol{v}_{i_t}$ are linearly independent is at least $\frac{3}{4}$. This implies, by Lemma 4, that the monomial π will survive from the replacements at Step ii with probability at least $\frac{3}{4}$. Furthermore, by expression (3) in Lemma 4,

$$
((p-1)\boldsymbol{v}_1+\boldsymbol{v}_0)^{m_i}\cdots((p-1)\boldsymbol{v}_t+\boldsymbol{v}_0)^{m_t}=\sum_{i=1}^{p^k}c(\pi)_i\boldsymbol{u}_i(\pi), \qquad (12)
$$

where $c(\pi)_i$ are coefficients in Z_p such that $(m_1+1)(m_2+1)\cdots(m_t+1)$ of them are nonzero, and $\boldsymbol{u}_i(\pi)$ are distinct vectors in Z_p^d. Let $\psi(\pi)$ be the product of the new variables y_j that are added with respect to the gates in C such that those gates produce the monomial π. Then, $\psi(\pi)$ is a monomial that is generated by C'. Hence, at Step iii, by expression (12) F' will have monomials respect to π as given in the following expansion:

$$\begin{aligned}\phi(\pi) &= \psi(\pi)\cdot((p-1)\boldsymbol{v}_1+\boldsymbol{v}_0)^{m_i}\cdots((p-1)\boldsymbol{v}_t+\boldsymbol{v}_0)^{m_t}\\ &= \sum_{i=1}^{p^k} c(\pi)_i\cdot\psi(\pi)\cdot\boldsymbol{u}_i(\pi).\end{aligned} \tag{13}$$

Let $\mathcal{S}$ be the set of all those p-monomials that survive from the variable replacements. Then,

$$\begin{aligned}&F'(y_1,\ldots,y_h,(\boldsymbol{v}_1+\boldsymbol{v}_0),\ldots,(\boldsymbol{v}_n+\boldsymbol{v}_0)) = \sum_{\pi\in\mathcal{S}}\phi(\pi)\\ &= \sum_{\pi\in\mathcal{S}}\left(\sum_{i=1}^{p^k} c(\pi)_i\cdot\psi(\pi)\cdot\boldsymbol{u}_i(\pi)\right)\\ &= \sum_{j=1}^{2^d}\left(\sum_{\pi\in\mathcal{S}\text{ and }\boldsymbol{z}_j=\boldsymbol{u}_i(\pi)} c(\pi)_i\cdot\psi(\pi)\right)\cdot\boldsymbol{z}_j\end{aligned} \tag{14}$$

Let

$$f_j(y_1,\ldots,y_h) = \sum_{\pi\in\mathcal{S}\text{ and }\boldsymbol{z}_j=\boldsymbol{u}_i(\pi)} c(\pi)_i\cdot\psi(\pi),$$

then the degree k polynomial with respect to $\boldsymbol{z}_j$ is obtained for F' in expression (10).

Recall that when constructing the circuit C', each new gate is associated with a new variable. This means that for any two monomials π' and π'' in F, we have $\psi(\pi')\neq\psi(\pi'')$. This implies that we cannot add $c(\pi')\cdot\psi(\pi')$ to $c(\pi'')\cdot\psi(\pi'')$ in f_j. Thus, the possibility of a "zero-sum" of coefficients from different surviving monomials is completely avoided during the construction of f_j. Therefore, conditioned on that $\mathcal{S}$ is not empty, F' must not be identical to zero, i.e., there exists at least one f_j that is not identical to zero. At Step iv, we use the randomized algorithm by Agrawal and Biswas [1] to test whether f_j is identical to zero. It follows from Theorem 4.6 in Agrawal and Biswas [1] that this testing can be done with probability at least $\frac{5}{6}$ in time polynomially in $s(n)$ and $\log q$. Since $\mathcal{S}$ is not empty with probability at least $\frac{3}{4}$, the probability of overall success of testing whether F has a p-monomial is at least $\frac{5}{8}$.

Finally, we address the issues about how to calculate F' and the time needed to do so. Naturally, every element in the group algebra $Z_p[Z_p^d]$ can be represented by a vector in $Z_p^{p^d}$. Adding two elements in $Z_p[Z_p^d]$ is equivalent to adding the two corresponding vectors in $Z_p^{p^d}$, and the latter can be done in $O(p^d\log p)$ time via component-wise sum. In addition, multiplying two elements in $Z_p[Z_p^d]$ is equivalent to multiplying the two corresponding vectors in $Z_p^{p^d}$, and the latter can be done in $O(dp^d \log^2 p)$ with the help of a similar Fast Fourier Transform style algorithm as in Williams [20]. Calculating F' consists of $s(n)$ arithmetic operations of either adding or multiplying two elements in $Z_p[Z_p^d]$ based on the circuit C or C'. Hence, the total time needed is $O(s(n)dp^d log^2 p)$. At Step iv, we run the Agrawal and Biswas [1] algorithm to F' to simultaneously test whether there is one f_j such that f_j is not identical to zero. We choose a probability $\frac{5}{6}$. By Theorem 4.6 in Agrawal and Biswas [1], this testing can be done in $O^*((s(n))^4 n^4 log^2 p)$ time, suppressing a poly$(\log s(n), \log n, \log\log p)$ factor. Recall that $d = k + log_p k + 1$. The total time for the entire algorithm is $O^*(p^k)$.

4 Derandomization

In this section, we turn our attention to formulas instead of general arithmetic circuits. We shall derandomize Steps i and iv in algorithm RT-MLM respectively with the help of two advanced techniques of perfect hashing by Chen *et al.* [8] (see also Naor *et al.* [17]) and noncommunicative multivariate polynomial identity testing by Raz and Shpilka [18].

Let n and k be two integers such that $1 \leq k \leq n$. Let $\mathcal{A} = \{1, 2, \ldots, n\}$ and $\mathcal{K} = \{1, 2, \ldots, k\}$. A k-coloring of the set $\mathcal{A}$ is a function from $\mathcal{A}$ to $\mathcal{K}$. A collection $\mathcal{F}$ of k-colorings of $\mathcal{A}$ is a (n, k)-family of *perfect hashing functions* if for any subset W of k elements in $\mathcal{A}$, there is a k-coloring $h \in \mathcal{F}$ that is injective from W to $\mathcal{K}$, i.e., for any $x, y \in W$, $h(x)$ and $h(y)$ are distinct elements in $\mathcal{K}$.

Theorem 2. *Let p be a prime number. Let $F(x_1, x_2, \ldots, x_n)$ be an n-variate polynomial of degree k represented by a formula C of size $s(n)$. There is a deterministic $O((6.4p)^k)$ time algorithm to test whether F has a p-monomial of degree k in its sum-product expansion.*

Proof. As in the proof of Theorem 1, we consider the group algebra $Z_p[Z_p^k]$. Here, we do not need to expand the dimension k to $d > k$. We also construct a new formula C' from C by adding new variable y_i for each multiplication gate g_i in the same way as what we did for Theorem 1. Assume that C has h many multiplication gates, then C' will have h new multiplication gates corresponding to new variables $y_1, y_2, \ldots, y_h$. The algorithm for testing whether F has a p-monomial of degree k is given as follows.

Algorithm DT-MLM (Deterministic Testing of Multilinear Monomials):

1. Construct with the algorithm by Chen *at el.* [8] an (n, k)-family of perfect hashing functions $\mathcal{H}$ of size $O(6.4^k \log^2 n)$.
2. Select k linearly independent vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in Z_p^k$. (No randomization is needed at this step.)
3. For each perfect hashing function $\tau \in \mathcal{H}$ do

a. For each variable x_i, replace it by $(\boldsymbol{v}_{\tau(i)} + \boldsymbol{v}_0)$.
b. Use C' to calculate

$$\begin{aligned} &F'(y_1, \ldots, y_h, (\boldsymbol{v}_1 + \boldsymbol{v}_0), \ldots, (\boldsymbol{v}_n + \boldsymbol{v}_0)) \\ &= \sum_{j=1}^{2^k} f_j(y_1, y_2, \ldots, y_h) \cdot \boldsymbol{z}_j, \end{aligned} \tag{15}$$

where each f_j is a polynomial of degree k over the finite field Z_p, and vectors $\boldsymbol{z}_j$ with $1 \le j \le 2^k$ are the 2^k distinct vectors in Z_p^k.
c. Perform polynomial identity testing with the Raz and Shpilka algorithm [18] for every f_j over Z_p. Stop and return *"yes"* if one of them is not identical to zero.

iv. If all perfect hashing functions in $\mathcal{H}$ have been tried without returning *"yes"*, then stop and output *"no"*.

By Chen *at el.*[8], Step i can be done in $O(6.4^k n \log^2 n)$ times. Step ii can be easily done in $O(k^2 \log p)$ time.

It follows from Lemma 3 that all those monomials that are not p-monomials in F, and hence in F', will be annihilated, when variables x_i are replaced by $(\boldsymbol{v}_i + \boldsymbol{v}_0)$ at Step iii.a.

Consider any given p-monomial $\pi = x_{i_1}^{m_1} \cdots x_{i_t}^{m_t}$ of degree k with $1 \le m_i < p$ and $k = m_1 + \cdots + m_t$, $i = 1, \ldots, t$. Because of the nature of $\mathcal{H}$, there is at least one perfect hashing function τ in $\mathcal{H}$ such that $\tau(i_{j'}) \neq \tau(i_{j''})$ if $i_{j'} \neq i_{j''}$, $1 \le j', j'' \le t \le k$. This means that $\boldsymbol{v}_{\tau(i_1)}, \ldots, \boldsymbol{v}_{\tau(i_t)}$ are distinct and hence linearly independent. By Lemma 4, π will survive from the replacements at Step iii.a. Let $\mathcal{S}$ be the set of all surviving p-monomials. Following the same analysis as in the proof of Theorem 1, we have F' that is not identical to zero if $\mathcal{S}$ is not empty. That is, there is at least one f_j that is not identical to zero, if $\mathcal{S}$ is not empty. Moreover, the time needed for calculating F' is $O(kp^k \log^2 p)$.

We now consider imposing noncommunicativity on C' as follows. Inputs to an arithmetic gate are ordered so that the formal expressions $y_{i_1} \cdot y_{i_2} \cdot \cdots \cdot y_{i_r}$ and $y_{j_1} \cdot y_{j_2} \cdot \cdots \cdot y_{j_l}$ are the same iff $r = l$ and $i_q = j_q$ for $q = 1, \ldots, r$. Finally, we use the algorithm by Raz and Shpilka [18] to test whether $f_j(y_1, \ldots, y_h)$ is identical to zero of not. This can be done in time polynomially in $s(n)$ and n, since f_j is a non-communicative polynomial represented by a formula.

Combining the above analysis, the total time of the algorithm DT-MLM is $O^*((6.4p)^k)$.

5 $\Pi_m \Sigma_2 \Pi_t \times \Pi_k \Sigma_3$ Polynomials

It has been proved by Chen and Fu [6] that the problem of testing monomials in $\Pi_m \Sigma_s$ (or $\Pi_m \Sigma_2 \Pi_t$) polynomials is solvable in $O(ms\sqrt{m+s})$ (or $O((mt)^2)$) time, but the problem for $\Pi_m \Sigma_3$ (or $\Pi_m \Sigma_2 \Pi_t \times \Pi_k \Sigma_3$) polynomials is NP-complete. Moreover, a $O(tm^2 1.7751^m)$ time algorithm was obtained for $\Pi_m \Sigma_3 \Pi_t$

polynomials, and so was a $O((mt)^2 3^k)$ algorithm for $\Pi_m \Sigma_2 \Pi_t \times \Pi_k \Sigma_3$ polynomials. In this section, we shall devise two parameterized algorithms, one deterministic and the other randomized, for testing multilinear monomials in $\Pi_m \Sigma_2 \Pi_t \times \Pi_k \Sigma_3$ polynomials, improving the $O((mt)^2 3^k)$ upper bound in [6].

Theorem 3. *There is a deterministic algorithm of time $O(((mt+k)^2+k)2^k)$ to test whether any $\Pi_m \Sigma_2 \Pi_t \times \Pi_k \Sigma_3$ polynomial has a multilinear monomial in its sum-product expansion.*

Proof. Let $F = F_1 \cdot F_2$ such that $F_1 = f_1 \cdots f_m$ is a $\Pi_m \Sigma_2 \Pi_t$ polynomial and $F_2 = g_1 \cdots g_k$ is a $\Pi_k \Sigma_3$ polynomial, where $f_i = (T_{i1} + T_{i2})$ and $g_j = (x_{j1} + x_{j2} + x_{j3})$, $1 \le i \le m$, $1 \le j \le k$.

Consider variable x_{11} in the clause g_1. We devise a branch and bound process to divide the testing for F into the testing for two new polynomials. We eliminate all x_{11} in g_j for $j = 1, \ldots, k$. Let g'_j be the clause resulted from g_j after the eliminating process. Let $h_1 = F_1 \cdot g'_1$, $h_2 = F_1 \cdot x_{11}$, $q = g'_2 \ldots g'_k$. Note that exactly one of the three variable x_{11}, x_{12} and x_{13} in the clause g_1 must be selected to form a monomial (hence a multilinear monomial) for F in the sum-product expansion of F. We have two cases concerning the selection of x_{11}:

(1) x_{11} can not be selected to help form any multilinear monomial. In this case, F has a multilinear monomial, iff $h_1 \cdot q$ has a multilinear monomial.

(2) x_{11} can be selected to form a multilinear monomial. Thus, F has a multilinear monomial, iff $h_2 \cdot q$ has a multilinear monomial.

In either case, the new polynomial is a product of two polynomials with the first being a $\Pi_{m+1} \Sigma_2 \Pi_t$ polynomial and the second a $\Pi_k \Sigma_3$ polynomial. Furthermore, the second is the common q, which has one fewer clause than F_2.

Let $T(k)$ denote the time for testing multilinear monomials in F. Notice that the eliminating process for x_{11} takes $O(k)$ time. Then, $T(k)$ is bounded as follows

$$T(k) \le 2T(k-1) + O(k) \le 2^k (T(0) + O(k)).$$

$T(0)$ is the time to test multilinear monomials in a $\Pi_{m+k} \Sigma_2 \Pi_t$ polynomial with a size of $O(mt+k)$. By the algorithm in [6] for this type of polynomials, $T(0) = O((mt+k)^2)$. Therefore, $T(k) = O(((mt+k)^2 + k)2^k)$.

We now show that the upper bound in the above theorem can be further improved via randomization.

Theorem 4. *There is a $O((mt+k)^2 1.5^k))$ time randomized algorithm that finds a multilinear monomial for any $\Pi_m \Sigma_2 \Pi_t \times \Pi_k \Sigma_3$ polynomial with probability at least $1 - \frac{1}{e}$ if such monomials exist, or returns* "no" *otherwise.*

Proof. Like in Theorem 3, let $F = F_1 \cdot F_2$ such that $F_1 = f_1 \cdots f_m$ is a $\Pi_m \Sigma_2 \Pi_t$ polynomial and $F_2 = g_1 \cdots g_k$ is a $\Pi_k \Sigma_3$ polynomial with $f_i = (T_{i1} + T_{i2})$ and $g_j = (x_{j1} + x_{j2} + x_{j3})$.

Assume that F has a multilinear monomial π. Then, one of the three variables in g_j must be included in π, $1 \le j \le k$. We uniformly select two distinct variables

y_{j1} and y_{j2} from g_j, then $g'_j = (y_{j1} + y_{j2})$ contains a desired variable for π with a probability at least 2/3. Let

$$F' = F_1 \cdot (g'_1 \cdots g'_k),$$

then F' has a multilinear monomial with a probability at least $(\frac{2}{3})^k$. On the other hand, if F does not have any multilinear monomials in its sum-product expansion, then F' must not have any multilinear monomials. Notice that F' is a $\Pi_{m+k}\Sigma_2\Pi_t$ polynomial with a size of $O(mt + k)$. By the algorithm for this type of polynomials by Chen and Fu in [6], one can find a multilinear monomial in F' in time $O((mt + k)^2)$. In other words, the above randomized process will fail to find a multilinear monomial in F with a probability of at most $1 - \left(\frac{2}{3}\right)^k$ if such monomials exist, or return *"no"* otherwise.

Repeat the above randomized process $\left(\frac{3}{2}\right)^k$ many times. If F has multilinear monomials, then these processes will fail to find one with a probability of at most

$$\left[1 - \left(\frac{2}{3}\right)^k\right]^{\left(\frac{3}{2}\right)^k} < \frac{1}{e}.$$

Hence, the processes will find a multilinear monomial in F with a probability of at least $1 - \frac{1}{e}$. If F does not have any multilinear monomial, then none of these repeated processes will find one in F. The total time of all the repeated processes is $O((mt + k)^2 1.5^k)$.

6 W[1]-Hardness

Although deterministic and randomized parameterized algorithms have been devised for testing monomials in previous three sections as well as in [14,20,6], yet we shall prove in this section that testing some special type of monomials in polynomials represented by arithmetic circuits is not fixed-parameter tractable, unless some unlikely collapse occurs in the fixed parameter complexity theory.

One shall notice that difference between the general monomial testing and the specific monomial testing. The former asks for the existence of *"any one"* from a set of possibly many monomials that are needed. The latter asks for *"a specific one"* from the set. For example, there may be $2^n - 1$ multilinear monomials in the sum-product expansion of a n-variate polynomials. Testing for any one from these many monomials is certainly different from testing for a specific one, say, $x_1x_3x_7x_{11}$.

Downey and Fellows [9] have established a hierarchy of parameterized complexity, named the W hierarchy, and proved that the k-Clique problem is W[1]-hard.

Definition 2. *Let $C = \{i_1, i_2, \ldots, i_k\}$ be a set of k positive integers. A k-clique monomial with respect to C is the multilinear monomial $\prod_{1 \le j < \ell \le k} x_{i_j i_\ell}$ of degree $\frac{k(k-1)}{2}$.*

Theorem 5. *It is W[1]-hard to test whether any given $n-$variate polynomial of degree $\frac{k(k-1)}{2}$ represented by an arithmetic circuit has a k-clique monomial in its sum-product expansion.*

Proof. We shall reduce the k-clique problem to the k-clique monomial testing problem. Let $G = (V, E)$ be an undirected graph and k an integer parameter. $V = \{v_1, v_2, \dots, v_m\}$ is the set of vertices. Each $(i, j) \in E$ represents the edge connecting vertices v_i and v_j. For each edge $(i, j) \in E$, we define a variable x_{ij}. Let $n = |E|$. We construct a polynomial f with n variables.

$$f(G,1) = 1,$$
$$f(G,2) = \sum_{(i,j)\in E} x_{ij},$$
$$f(G,t+1) = \sum_{i=1}^{m} \left(\sum_{(i,j)\in E} x_{ij} \right)^{t} \cdot f(G,t)$$

As followed from the above definition, $f(G,k)$ has $n = |E|$ variables and its degree is $\frac{k(k-1)}{2}$. It is easy to see that $f(G,k)$ can be computed by an arithmetic circuit.

If G has a k-clique $A = \{i_1, i_2 \dots, i_k\}$, then there are $\frac{k(k-1)}{2}$ edges connecting any two vertices in A. By definition, $f(G,k)$ has a term $(x_{i_1 i_2} + \cdots + x_{i_1 i_k} + \cdots + x_{i_{k-1} i_k})^{k-1} \cdot f(G, k-1)$. So, we can select $\pi_1 = x_{i_1 i_2} \cdots x_{i_1 i_k}$ from the first factor of this term. By simple induction, we can select a $(k-1)$-clique monomial of degree $\frac{(k-1)(k-2)}{2}$ with respect to $A - \{i_1\}$. Then, $\pi_1 \cdot \pi_2$ is a k-clique monomial with respect to A. On the other hand, it $f(G,k)$ has a k-clique monomial with respect to A, then by definition, A is a k-clique for G.

References

1. Agrawal, M., Biswas, S.: Primality and identity testing via Chinese remaindering. Journal of the ACM 50(4), 429–443 (2003)
2. Agrawal, M., Kayal, N., Saxena, N.: PRIMES is in P. Ann. of Math. 160(2), 781–793 (2004)
3. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. Journal of the ACM 45(3), 501–555 (1998)
4. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters 8(3), 121–123 (1979)
5. Beigel, R.: The polynomial method in circuit compplexity. In: Proceedings of the Eighth Conference on Structure in Complexity Theory, pp. 82–95 (1993)
6. Chen, Z., Fu, B.: The complexity of testting monomials in multivariate polynomials. In: Proceedings of the Fifth International Conference on Combinatorial Optimization and Applications (2011)
7. Chen, Z., Fu, B.: Approximating multilinear monomial coefficients and maximum multilinear monomials in multivariate polynomials. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part I. LNCS, vol. 6508, pp. 309–323. Springer, Heidelberg (2010) (The full version will appear in Journal of Combinatorial Optimization)

8. Chen, J., Lu, S., Sze, S.-H., Zhang, F.: Improved algorithms for path, matching, and packing problems. In: SODA, pp. 298–307 (2007)
9. Downey, R.G., Fellows, M.R.: Fixed parameter tractability and completeness. II. On completeness for W(1). Theoretical Computer Science 141(1-2), 109–131 (1995)
10. Feige, U., Goldwasser, S., Lovász, L., Safra, S., Szegedy, M.: Interactive proofs and the hardness of approximating cliques. Journal of the ACM 43(2), 268–292 (1996)
11. Fu, B.: Separating PH from PP by relativization. Acta Math. Sinica 8(3), 329–336 (1992)
12. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. In: STOC, pp. 355–364 (2003)
13. Klivans, A., Servedio, R.A.: Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. In: STOC, pp. 258–265 (2001)
14. Koutis, I.: Faster algebraic algorithms for path and packing problems. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 575–586. Springer, Heidelberg (2008)
15. Minsky, M., Papert, S.: Perceptrons (expanded edition 1988). MIT Press, Cambridge (1968)
16. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
17. Naor, M., Schulman, L.J., Srinivasan, A.: Splitters and near-optimal derandomization. In: FOCS, pp. 182–191 (1995)
18. Raz, R., Shpilka, A.: Deterministic polynomial identity testing in non-commutative models. Computational Complexity 14(1), 1–19 (2005)
19. Shamir, A.: IP = PSPACE. Journal of the ACM 39(4), 869–877 (1992)
20. Williams, R.: Finding paths of length k in $O^*(2^k)$ time. Information Processing Letters 109, 315–318 (2009)

Hybrid Artificial Bee Colony Search Algorithm Based on Disruptive Selection for Examination Timetabling Problems

Malek Alzaqebah and Salwani Abdullah

Data Mining and Optimisation Research Group (DMO),
Center for Artificial Intelligence Technology,
Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor, Malaysia
{malek_zaqeba,salwani}@ftsm.ukm.my

Abstract. Artificial Bee Colony (ABC) is a population-based algorithm that employed the natural metaphors, based on foraging behavior of honey bee swarm. In ABC algorithm, there are three categories of bees. Employed bees select a random solution and apply a random neighborhood structure (exploration process), onlooker bees choose a food source depending on a selection strategy (exploitation process), and scout bees involves to search for new food sources (scouting process). In this paper, firstly we introduce a disruptive selection strategy for onlooker bees, to improve the diversity of the population and the premature convergence, and also a local search (i.e. simulated annealing) is introduced, in order to attain a balance between exploration and exploitation processes. Furthermore, a self adaptive strategy for selecting neighborhood structures is added to further enhance the local intensification capability. Experimental results show that the hybrid ABC with disruptive selection strategy outperforms the ABC algorithm alone when tested on examination timetabling problems.

Keywords: Artificial Bee Colony, Simulated Annealing, Examination Timetabling Problems, Disruptive Selection.

1 Introduction

The examination timetabling problem is concerned with allocating a set of examinations into a limited number of timeslots (periods), subject to a set of constraints. The basic challenge of examination timetabling is to schedule examinations over a limited number of timeslots, so as to avoid conflicts (refer to hard constraints) and to satisfy a number of side constraints (refer to soft constraints) [5]. In recent years, a variety of constraints has been addressed in the scientific literature to model the real-world problems and try to close the gap between theory and practice in automated timetabling, as presented by the 2nd International Timetabling Competition (ITC2007) where three tracks of problems are proposed i.e. one for exam timetabling and two for course timetabling.

In the past, a wide variety of approaches for solving the examination timetable problem have been described and discussed in the literature, that basically can be divided into population-based approaches and single solution based approaches. For a recent detailed overview readers should consult [8].

In this paper, we employ a population-based approach that is based on a swarm intelligence algorithm, called artificial bee colony (ABC). It was proposed by Karaboga [1]. ABC algorithm has been successfully developed to solve many optimisation problems [18, 13, and 19]. It mimics the foraging behavior of honey bee swarms. This work concentrates incorporation of the ABC algorithm with a local search (a simulated annealing in this case) to compensate for the insufficiency of using each type of method in isolation. The details of the proposed approach are discussed later.

The paper is organised as follows. Section 2 presents the examination timetabling problem and its formulation. The original Artificial Bee Colony algorithm is presented in Section 3. The proposed approach is discussed in Section 4. Our experimental results are presented in Section 5. This is followed by conclusion and comments in Section 6.

2 Problem Description and Formulation

The problem description in this paper is divided into two parts as discussed below:

- Problem I: This problem is introduced by Carter et al. (1996) [5], which considered as an uncapacitated examination timetabling problem, where a room capacity requirement is not taking into account.
- Problem II: International timetabling competition (ITC2007) datasets which consist of three tracks. In this work, we consider the first track that represents an exam timetabling model which includes a number of real world constraints.

2.1 Problem I

The problem description that is utilised in this paper is adapted from the description presented in Burke et al. (Burke et al., 2004). Examination timetabling problems consist of these inputs as stated below:

- N is the number of exams
- E_i is an exam, $i \in \{1,\ldots,N\}$
- T is the given number of available timeslots
- M is the number of students
- $C = (c_{ij})_{NxN}$ is the conflict matrix where each element denoted by c_{ij}, $i,j \in \{1,\ldots,N\}$ is the number of students taking exams i and j.
- t_k $(1 \leq t_k \leq T)$ specifies the assigned timeslot for exam k ($k \in \{1,\ldots,N\}$)

We formulate an objective function which tries to space out students' exams throughout the exam period (Expression (1)) that can then be formulated as the minimisation of:

$$\frac{\sum_{i=1}^{N-1} F_1(i)}{M} \tag{1}$$

where

$$F_1(i) = \sum_{j=i+1}^{N} c_{ij} \cdot proximity \quad (t_i, t_j) \tag{2}$$

and

$$proximity \quad (t_i, t_j) = \begin{cases} 2^5 / 2^{|t_i - t_j|} & if \quad 1 \le |t_i - t_j| \le 5 \\ 0 & otherwise \end{cases} \tag{3}$$

subject to:

$$\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij} \cdot \lambda(t_i, t_j) = 0$$

where

$$\lambda(t_i, t_j) = \begin{cases} 1 & if \quad t_i = t_j \\ 0 & otherwise \end{cases} \tag{4}$$

Equation (2) presents the cost for an exam *i* which is given by the proximity value multiplied by the number of students in conflict. Equation (3) represents a proximity value between two exams [5]. Equation (4) represents a clash-free requirement so that no student is asked to sit two exams at the same time. The clash-free requirement is considered to be a hard constraint.

2.2 Problem II

International timetabling competition (ITC2007) introduces three tracks of problems i.e. examination timetabling, curriculum-based course timetabling problem and post-enrolment course timetabling problems. In this paper, we concentrate on the first track i.e. examination timetabling problems that include a number of real world constraints [8]. A set of hard and soft constraints are listed in Table 1 and Table 2, respectively.

Table 1. Hard Constraints

Hard Constraints	Explanation
H1	There cannot be any students sitting for more than one exam at the same time.
H2	The total number of students assigned to each room cannot exceed the room capacity.
H3	The length of exams assigned to each timeslot should not violate the timeslot length.
H4	Some sequences of exams have to be respected. e.g. Exam_A must be schedule after Exam_B.
H5	Room related hard constraints must be satisfied e.g. Exam_A must be scheduled in Room 80.

Table 2. Soft Constraints

Soft Constraints	Mathematical Symbol	Explanation
S1	C_S^{2R}	Two exams in a row: Minimise the number of consecutive exams in a row for a student.
S2	C_S^{2D}	Two exams in a day: student should not be assigned to sit more than two exams in a day. Of course, this constraint only becomes important when there are more than two examination periods in the same day.
S3	C_S^{PS}	Periods spread: all students should have a fair distribution of exams over their timetable.
S4	C_S^{2NMD}	Mixed durations: The numbers of exams with different durations that are scheduled into the same room has to be minimised as much as possible.
S5	C^{FL}	Larger examinations appearing later in the timetable: Minimise the number of examinations of large class size that appear later in the examination timetable (to facilitate the assessment process)
S6	C^{P}	Period Penalty: some periods have an associated penalty, minimise the number of exams scheduled in penalised periods.
S7	C^{R}	Room Penalty: some rooms have an associated penalty, minimise the number of exams scheduled in penalised rooms.

A feasible timetable is one in which all examinations have been assigned to a period and room, and there is no violation of hard constraints. The objective function is to minimise the violation of soft constraints as given in expression (5) [8].

$$\min \sum_{s\in S} (w^{2R} C_S^{2R} + w^{2D} C_S^{2D} + w^{PS} C_S^{PS}) + w^{NMD} C_S^{2NMD} + w^{FL} C^{FL} + w^{p} C^{P} + w^{R} C^{R} \quad (5)$$

Each dataset has its own weight as shown in Table 3 [8].

Table 3. The Associate Weight of ITC2007 Collection of Examination Datasets

Datasets	w^{2}	w^{2}	$w^{P.}$	w^{N}	w^{FL}	w^{F}	w^{R}
Exam_1	5	7	5	10	100	30	5
Exam_2	5	15	1	25	250	30	5
Exam_3	10	15	4	20	200	20	10
Exam_4	5	9	2	10	50	10	5
Exam_5	15	40	5	0	250	30	10
Exam_6	5	20	20	25	25	30	15
Exam_7	5	25	10	15	250	30	10
Exam_8	0	150	15	25	250	30	5

3 Artificial Bee Colony Algorithm (ABC)

3.1 Basic Artificial Bee Colony (ABC) Algorithm

Artificial Bee Colony (ABC) algorithm is a global optimisation algorithm that replicates the real behavior of honey bees, introduced by Karaboga [1]. The algorithm classifies the bees in the hive into three groups i.e. employed bees, onlooker bees and scouts bees. In this algorithm, employed bees fly around the search space to collect the information of food sources, and share the information with onlooker bees through a wiggle dance. Then onlooker bees choose their food sources based on this information. During the search process, the employed bees whose food source has been abandoned become scout bees and start to search for new food sources randomly without any previous information. In ABC algorithm, the position of a food source represents a possible solution, and the nectar amount of a food source corresponds to the quality (fitness value) of the associated solution. The number of the employed bees is tied to the number of solutions in the population. Figure 1 shows the pseudo code of the ABC algorithm [1].

```
Initial food sources are produced for all employed bees
REPEAT
   Each employed bee fly's to a food source in her memory and
        determines a neighbour source, then evaluates its nectar
        amount and dances in the hive
   Each onlooker watches the dance of employed bees and chooses one
        of their sources depending on the dances, and then goes to that
        source. After choosing a neighbour around that, she evaluates
        its nectar amount.
   Abandoned food sources are determined and are replaced with the
        new food sources discovered by scouts.
   The best food source found so far is registered.
UNTIL (requirements are met)
```

Fig. 1. Original artificial bee colony search algorithm

As shown in Figure 1, at the first step, a constructive heuristic algorithm is applied to initialise the population (food source positions). After the initialization, an employed bee produces an adjustment on the source position in her memory and discovers a new food source position. If the nectar amount of the new food source is higher than the previous food source, the bee memorises the new source position and forgets the old one. Otherwise, she keeps the position of the one in her memory. After all the employed bees complete the search process, they share the information of the sources with the onlookers on the dance area. Each onlooker bee evaluates the nectar information taken from all employed bees and then chooses a food source depending on the nectar amounts. Finally scout bees find out the abandoned sources and replace it by randomly produced sources.

3.2 Onlooker Bees Selection Process

Onlooker bees choose the solution by a stochastic selection strategy, which is summarised as below:

1. Calculates the fitness value (fit_i) by using the fitness function as follow:

$$fit_i = \begin{cases} \dfrac{1}{1+f_i} & f_i \geq 0 \\ 1 + abs(f_i) & f_i < 0 \end{cases} \tag{6}$$

   where f_i is fitness function.

2. Calculate the probability value by using the following expression:

$$p_i = \frac{f_i}{\sum_{i=1}^{SN} f_i} \tag{7}$$

   where SN is the number of food sources, f_i is the fitness function of the i^{th} food source.
3. Finally, chose a candidate solution based on the selection probability by "roulette wheel selection" method.

As stated in [10], there are two problems of using basic ABC selection strategy i.e. (i) A "super- individual" being too often selected the whole population tends to converge towards his position. The diversity of the population is then too reduced to allow the algorithm to progress; (ii) with the progression of the algorithm, the differences between fitness are reduced. The best ones then get quite the same selection probability as the others and the algorithm stops progressing." Thus, this selection strategy is hard to keep the diversity and to avoid the premature convergence. In order to alleviate these problems, this paper employed a disruptive selection strategy to improve the performance of the ABC algorithm.

3.3 Disruptive Selection Strategy

Disruptive selection gives more chances for higher and lower individuals to be selected by changing the definition of the fitness function as in Equation (8) [13].

$$fit_i = | f_i - \overline{f_t} | \qquad P_i \frac{fit_i}{\sum_{i=0}^{n} fit_i} \tag{8}$$

Where f_i is the fitness function, $\overline{f_t}$ is the average value of the fitness function f_i of the individuals in the population. By using a disruptive selection, it has a tendency to maintain the diversity slightly longer, because both higher and lower quality of the solutions is more preferable.

4 The Proposed Algorithm

4.1 Neighborhood Search Operations

In this paper, four neighborhood search operations are employed in order to enhance the performance of searching algorithms i.e. [2]:

Nbs1: Select 2 exams at random and swap timeslots.
Nbs2: Select a single exam at random and move to a new random feasible timeslots.
Nbs3: Select 4 exams randomly and swap the timeslots between them feasibly.
Nbs4: Select 2 exams at random and move to a new random feasible timeslots.

4.2 Self-adaptive Method for Neighbouring Search

To find neighbouring food sources, both employee and onlooker bees apply a self-adaptive strategy that is explained as follows [21]:

1. At the beginning, the neighbour list (*NL*) with a specified length is generated by filling the list randomly from four neighbourhood search operations as explained in Section 4.1.
2. During the evolution process, one neighbourhood is taken from *NL* and is used to generate a new food source for an employed bee or onlooker bee.
3. If the new food source is better than the current one, this approach will put the employed neighbourhood search operation into a new list, called a winning neighboring list (*WNL*).
4. When the *NL* became empty, it will be refilled as follow: 75% is refilled from the *WNL* list, and 25% is refilled randomly from four neighbourhood search operations, and also *WNL* is reset to zero to keep away of any accumulation effects.
5. If the *WNL* is empty (this perhaps happen when the search perform near an optimal with negligible population variety) the most recent *NL* is used again.

By using this method, the suitable neighboring search operation can be learned based on the current search process and the solution state. The length of *NL* is set to 200 as stated in [21].

4.3 A Local Search Algorithm (Simulated Annealing)

A simple local search (i.e. simulated annealing) is embedded to the basic ABC algorithm in order to enhance the utilisation capability of the algorithm. This is due to the basic ABC that uses a greedy acceptance criterion i.e. only accepts an improved solution and eliminate the worse.

Simulated annealing has been proposed by Kirkpatrick et al. [12]. It mimics the annealing process of metals molten that is heated and then slowly cooled. A simulated annealing algorithm works on a single solution and tries to improve it by finding nearby solutions, and slowly amending a parameter called temperature. The algorithm always accepts a better solution. In SA, a worse solution is accepted with the a certain probability between [0,1] if it less than $e^{-\delta/Temp}$ where δ is the difference between the penalty cost of the new and current solutions (i.e. $\delta = f(Sol^*)-f(Sol)$). The process is repeated until the temperature *Temp* is less than the final temperature T_f , as shown in Figure 2. In this paper, the parameter used for the simulated annealing algorithm are set as follow (adapted from Abdullah and Burke [20]): the initial temperature (T_0) is set to 5000, final temperature (T_f) is set to 0.05, and the number of iterations, $NumOfIte_{SA}$ is set to 200000.

4.4 Constructive Heuristic

In this paper, we use a graph colouring approach (i.e. largest degree heuristic) to generate the initial solution, where the examination with the largest number of conflicts are scheduled first. For more details about graph colouring applications to timetabling see Burke et al. [3]

4.5 Improvement Algorithm

Figure 2 illustrates the pseudo-code that represents our approach, and also a SA pseudo-code. The algorithm starts with feasible initial solutions which are generated by a largest degree heuristic.

The algorithm starts with initial population that is generated using a graph colouring approach. The employed bees work on random solutions and apply a neighborhood structure based on self-adaptive method (as explained in Section 4.2) on each solution. The solutions are arranged based on the profitability. Onlooker bees calculate the selection probability based on disruptive selection as in Equation (8) and then she applies a local search (SA) (as explained in Section 4.3) on the highest probability solution. Finally, scout bees determine the abandoned food source and replace it with the new food source.

```
Initialisation:
Initialise the initial population and evaluate the fitness;
Calculate the initial fitness value, f(Sol);
Set best solution, Solbest ← Sol;
Set maximum number of iteration, NumOfIte;
Set the population size;
//where population size = OnlookerBee = EmployeedBee;

iteration ←  0;
Improvement:
do while (iteration < NumOfIte)
  Exploration process
  for i=1: EmployeedBee
      Select a random solution and apply neighborhood
        structure based on Self-Adaptive;
  end for

  Exploitation process
  for i=1: OnlookerBee
    Calculate the selection probability $P_i$, based on
    disruptive selection as in Equation (8).
    Sol* ← select the solution depending on $P_i$;
    Start local search(SA) on Sol*;
      Set initial temperature $T_0$;
      Set final temperature $T_f$;
      Set number of iteration $NumOfIte_{SA}$;
      Set decreasing temperature rate as $\alpha$
      where $\alpha = (\log(T_0) - \log(T_f))/NumOfIte_{SA}$;
      Set Temp ← $T_0$;
      Set $Solbest_{SA}$ ← Sol*;
      Set $Sol_{SA}$ ← Sol*;
      do while (Temp > $T_f$)
       $Sol_{SA}*$ ← Apply neighbourhood structure on $Sol_{SA}$;
       Calculate cost function f($Sol_{SA}*$);
       if (f($Sol_{SA}*$) < f($Solbest_{SA}$))
         $Sol_{SA}$ ← $Sol_{SA}*$;
         $Solbest_{SA}$ ← $Sol_{SA}*$;
       else
         Generate a random number called RandNum;
         if (RandNum ≤ $e^{-\delta/Temp}$) where $\delta = f(Sol_{SA}*) - f(Sol_{SA})$
           $Sol_{SA}$ ← $Sol_{SA}*$;
           Temp = Temp-Temp*$\alpha$;
      end while
    end local search
    if (f($Solbest_{SA}$) < f(Sol*))
      Sol* ← $Solbest_{SA}$;
    end if
  end for
    Solbest ← best solution found so far;
  Scouting process
  Scoutbee determines the abandoned food source and replace
  it with the new food source.
  iteration++;
end do
```

Fig. 2. The pseudo code for the artificial bee colony search algorithm

5 Simulation Results

In our experimental results we employed three different modifications of ABC algorithm, called ABC algorithm based on disruptive selection (coded as DABC), DABC algorithm with a local search (i.e. Simulated Annealing) (coded $DABC_{SA}$) and $DABC_{SA}$ algorithm with a self-adaptive method for neighbouring search (coded as self-adaptive $DABC_{SA}$). We compare the performance of these modifications with the basic ABC in order to show the effects of employing different modification on basic ABC algorithm. The parameter settings used in this work are shown in Table 4.

Table 4. Parameters setting

Parameter	Value
Iteration	500
population size	50
Scout Bee	1

5.1 Problem I

Table 5 provides the comparison of basic ABC, DABC, $DABC_{SA}$ and self-adaptive $DABC_{SA}$ results, and compared with the best known results in the literature. The purpose here is to compare the performance among the three versions of ABC algorithms when tested in Problem I which represents the uncapacitated examination timetabling problem.

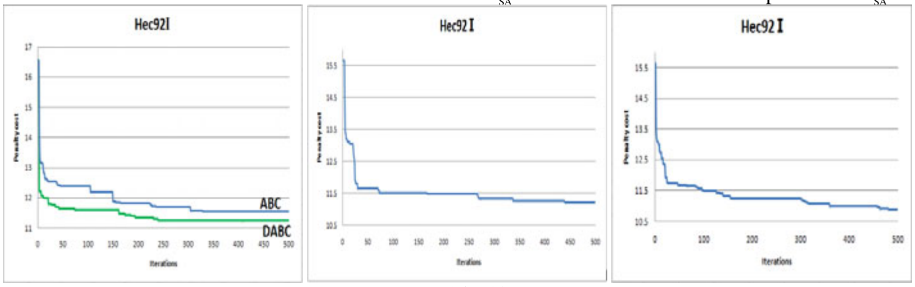
Table 5. Results Comparison on Uncapacitated Problems

Instance	Basic ABC	DABC	$DABC_{SA}$	Self adaptive $DABC_{SA}$	Best known	Authors for best known
car91	5.86	5.42	5.33	5.19	**4.50**	Yang and Petrovic (2004)
car92	4.92	4.84	4.39	4.36	**3.98**	Yang and Petrovic (2004)
ear83 I	38.34	37.54	35.17	32.26	**29.3**	Caramia et al. (2001)
hec92 I	11.51	11.21	11.19	10.89	**9.2**	Caramia et al. (2001)
kfu93	16.04	15.13	14.07	13.73	**13.0**	Burke et al. (2010)
lse92	12.42	12.06	11.89	11.15	**9.6**	Caramia et al. (2001)
sta83 I	158.12	157.52	157.39	157.23	**156.9**	Burke et al. (2010)
tre92	9.58	9.23	9.41	9.22	**7.9**	Burke et al. (2010)
uta92 I	3.99	3.94	3.89	3.83	**3.14**	Yang and Petrovic (2004)
ute92	27.80	27.57	27.11	26.73	**24.8**	Burke et al. (2010)
yor83 I	41.44	40.94	40.76	40.63	**34.9**	Burke et al. (2010)

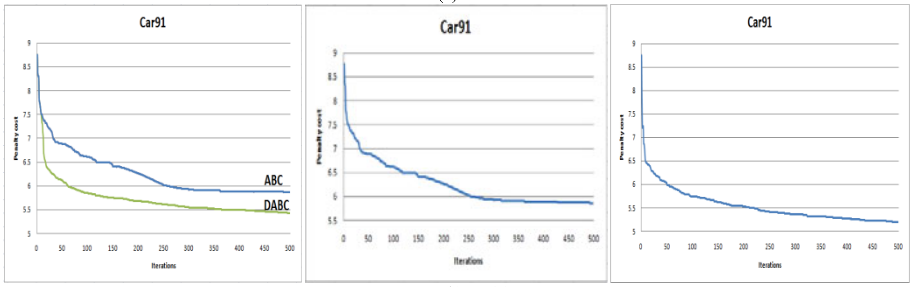
The comparison between basic ABC, DABC, $DABC_{SA}$ and the self-adaptive $DABC_{SA}$ shows, that the three modified version of ABC perform much better than the basic ABC. From Table 5, we can say that the disruptive selection strategy outperform the basic ABC, and then after applying the local search on DABC ($DABC_{SA}$) the algorithm is able to produce better solutions. The comparison between the $DABC_{SA}$ and the self-adaptive $DABC_{SA}$ shows that, by employing a self-adaptive method for neighbouring search helps the algorithm to perform better than with the local search alone (i.e. that select the neighbourhood search operations at random).

Overall comparison with the best known results shows that even we are unable to beat any of the best known results in the literature, we are still able to produce good enough solutions.

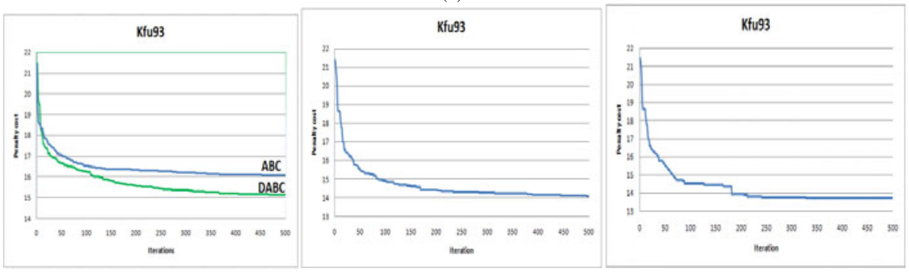
ABC+DABC DABC$_{SA}$ self-adaptive DABC$_{SA}$



(a) hec92I



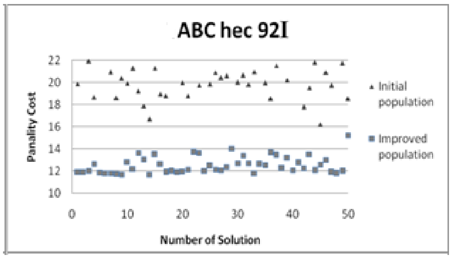
(b) car91



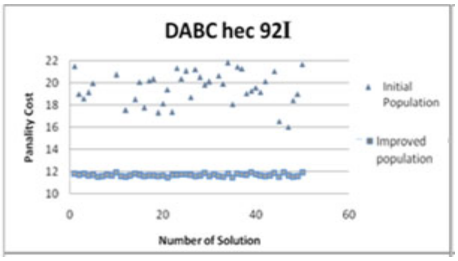
(c) kfu93

Fig. 3. Convergence graph for hec92I, car91 and kfu93

Figure 3 shows the behavior of the algorithm over three datasets. The *x*-axis represents the number of iterations, while the *y*-axis represents the penalty cost. The first column is the basic ABC convergence, the middle is the DABC$_{SA}$, and the right column is the self-adaptive DABC$_{SA}$. These graphs show how our algorithm explores the search space in which we believe that the way the algorithm behaves has a correlation with the complexity of the datasets (represented by the conflict density value). Note that the details of the conflict density values can be found in Qu et al. (2009). The higher conflict density signifies that more exams are conflicting with each other. The conflict density value for hec92I is 0.42, car91 is 0.13 and kfu93 is 0.06. The behavior of the algorithm works similar at the beginning of the iterations where the improvement of the solution can easily be obtained. Later it becomes steady and hard to be improved. However, for the kfu93 dataset (where the conflict density value is low compared to hec92I and car91 datasets), the algorithm is able to slowly improve the quality of the solution until it get stuck in the local optimum when the number of iteration almost reaches the maximum number of iteration used in this experiment.



(a)



(b)

Fig. 4. The effects of using the disruptive selection strategy

Figure 4 shows the effects of using the disruptive selection strategy (as explained in Section 3.3). The x-axis represents the number of solution, while the *y*-axis represents the penalty cost. The graph shows that DABC can explore the search space better than basic ABC. This is due to the behavior of the selection strategy i.e. in the basic ABC, a random selection is used to select the solution (as explained in Section 3.2) where the solution with highest fitness value will be the most selected during the improvement process. However, the disruptive selection strategy concentrates on both worse and high fitness solutions, and trying to keep the population diversity by improving the worse fitness solutions in concurrency with the high fitness solutions. This can be seen in Figure 4 (b) where all the solutions (improved population) are converged together after the improvement step are executed.

5.2 Problem II

The three modified version of ABC algorithm are also tested on Problem II which represents the (ITC2007). The results are shown in Table 6, which provides the comparison between the basic ABC, DABC, DABC$_{SA}$ and self-adaptive DABC$_{SA}$, and with other available results in the literature.

As shown in Table 6, the results for the three modified version of ABC (DABC, DABC$_{SA}$ and the self-adaptive DABC$_{SA}$) perform better than the basic ABC. The efficiency of the algorithm increases starting with applying a disruptive selection (DABC) and then the hybridization with a local search (DABC$_{SA}$). Form Table 6, we can conclude that the self-adaptive DABC$_{SA}$ shows better result in comparison with the three version of ABC, and they are comparable with some of other results in the literature.

Table 6. Results Comparison on ITC2007 Datasets

Datasets	Müller (2009)	Atsuta et al. (2007)	Pillay (2007)	Gogos et al. (2009)	ABC	DABC	DABC$_{SA}$	Self-adaptive DABC$_{SA}$
Exam_1	4370	8006	12035	4699	6582	6552	6361	6354
Exam_2	400	3470	3074	385	1517	1455	1377	1352
Exam_3	10049	18622	15917	8500	11912	11441	10867	10146
Exam_4	18141	22559	23582	14879	19657	19591	18929	18214
Exam_5	2988	4714	6860	2795	17659	17104	16942	16124
Exam_6	26950	29155	32250	25410	26905	23309	21309	22309
Exam_7	4213	10473	17666	3884	6840	6747	6692	6317
Exam_8	7861	14317	16184	7440	11464	10948	10857	10293

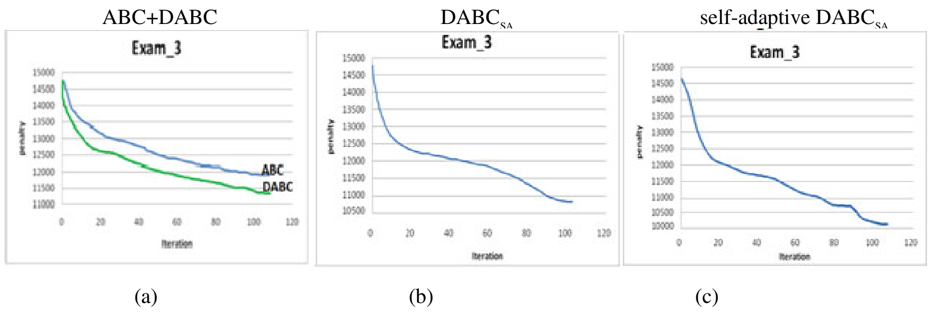


Fig. 5. Convergence graph for Exam_3

Figure 5 shows the behavior of the algorithm when tested on Problem II where the conflict density for Exam_3 is equal to 2.62 [15]. The *x*-axis represents the number of iterations, while the *y*-axis represents the penalty cost. As in Figure 5 (a, b and c), all of three modified algorithms are better than the basic ABC (based on the results obtained). Figure 5(a) shows that the DABC algorithm with disruptive selection strategy able to better explore the search space and brings all solutions to converge together. By incorporating the simulated annealing algorithm with DABC algorithm create a balance of exploration and exploitation. This is evidenced from the obtained results where the local search helps to further improve the obtained solutions. Introducing a self-adaptive strategy (compare to random) in selecting neighbourhood structures is managed to enhance the local intensification capability as shown in Figure 5(c). The results from Table 6 also show that the self-adaptive DABC$_{SA}$ is able to obtain solutions that are better than other proposed approaches here on almost of the tested datasets. Note that, the experiment carried out here is terminated when the time reach to 600 seconds (as set in the ITC2007 computation rules).

6 Conclusion and Future Work

The primary aim of this paper is to enhance the performance of the basic ABC algorithm by using a disruptive selection strategy (DABC), and later incorporate with a local search algorithm (i.e. a simulated annealing in this case) ($DABC_{SA}$). Adding a self adaptive strategy (self-adaptive $DABC_{SA}$) for selecting a neighborhood structure helps to further enhance the performance of $DABC_{SA}$. Experimental results show that the three modified version of ABC algorithm outperforms the ABC algorithm alone and are comparable with state-of-the-art when tested on examination timetabling problems. As a future work, we will investigate the effect of adaptively incorporating a different local search algorithm and tested its performance on broader timetabling problems.

References

1. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department (2005)
2. Abdullah, S., Ahmadi, S., Burke, E.K., Dror, M.: Investigating Ahuja-Orlin's large neighbourhood search approach for examination timetabling. OR Spectrum 29(2), 351–372 (2007)
3. Burke, E.K., Bykov, Y., Newall, J.P., Petrovic, S.: A time-predefined local search approach to exam timetabling problem. IIE Transactions 36(6), 509–528 (2004)
4. Caramia, M., Dell'Olmo, P., Italiano, G.F.: New algorithms for examination timetabling. In: Näher, S., Wagner, D. (eds.) WAE 2000. LNCS, vol. 1982, pp. 230–241. Springer, Heidelberg (2001)
5. Carter, M.W., Laporte, G.: Examination Timetabling: Algorithmic Strategies and Applications. Journal of the Operational Research Society 47, 373–383 (1996)
6. Carter, M.W.: A survey of practical applications of examination timetabling algorithms. Operations Research 34(2), 193–202 (1986)
7. Lewis, R.: A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum 30(1), 167–190 (2008)
8. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G.: A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12, 55–89 (2009)
9. Abdullah, S., Turabeih, H., McCollum, B.: A hybridization of electromagnetic like mechanism and great deluge for examination timetabling problems. In: Blesa, M.J., Blum, C., Di Gaspero, L., Roli, A., Sampels, M., Schaerf, A. (eds.) HM 2009. LNCS, vol. 5818, pp. 60–72. Springer, Heidelberg (2009)
10. Bao, L., Zeng, J.: Comparison and Analysis of the Selection Mechanism in the Artificial Bee Colony Algorithm. HIS 1, 411–416 (2009)
11. Abdullah, S., Burke, E.K., McCollum, B.: Using a Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for University Course Timetabling. In Metaheuristics: Progress in complex systems optimization (Operations Research / Computer Science Interfaces Series). Ch. 8. Springer, Heidelberg (2007) ISBN:978-0-387-71919-1

12. Camazine, S., Deneubourg, J.-L., Franks, N., Sneyd, J., Theraulaz, G.: Bonabeau.Self-Organization in Biological Systems. Princeton University Press, Princeton (2003)
13. Karaboga, N.: A new design method based on artificial bee colony algorithm for digital IIR filters. Journal of the Franklin Institute 346(4), 328–348 (2009)
14. Karaboga, N., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Applied Soft Computing 8, 687–697 (2008)
15. Karaboga, D., Akay, B.: A comparative study of artificial bee colony algorithm, Applied Mathematics and Computation (2009) doi:10.1016/j.amc.2009.03.90
16. Karaboga, D.: An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey (2005)
17. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization 39, 459–471 (2007)
18. Kang, F., Li, J., Xu, Q.: Structural inverse analysis by hybrid simplex artificial bee colony algorithms. Computers and Structures 87, 861–870 (2009)
19. Singh, A.: An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem. Applied Soft Computing 9, 625–631 (2009)
20. Abdullah, S., Burke, E.K.: A Multi-start large neighbourhood search approach with local search methods for examination timetabling. In: Long, D., Smith, S.F., Borrajo, D., McCluskey, L. (eds.) The International Conference on Automated Planning and Scheduling (ICAPS 2006), Cumbria, UK, June 6-10, pp. 334–337 (2006)
21. Pan, Tasgetiren, Q.-K., Suganthan, M.F., N., P., Chua, T.J.: A Discrete Artificial Bee Colony Algorithm for the Lot-streaming Flow Shop Scheduling Problem, Information Sciences. Elsevier, Netherlands (2010)
22. Burke, Eckersley, E.K., J. A., McCollum, B., Petrovic, S., Qu, R.: Hybrid variable neighbourhood approaches to university exam timetabling. European Journal of Operation Research 206(1), 46–53 (2010)
23. Yang, Y., Petrovic, S.: A novel similarity measure for heuristic selection in examination timetabling. In: Burke, E.K., Trick, M.A. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 247–269. Springer, Heidelberg (2005)

Heuristics for Parallel Machine Scheduling with Deterioration Effect

Ming Liu[1], Feifeng Zheng[2,3,4], Yinfeng Xu[2,3,4], and Lu Wang[5]

[1] School of Economomics & Management Tongji University, Shanghai, 200092, P.R. China
[2] School of Management, Xi'an JiaoTong University, Xi'an, Shaanxi, 710049, P.R. China
[3] The State Key Lab for Manufacturing Systems Engineering, Xi'an, Shaanxi, 710049, P.R. China
[4] Ministry of Education Key Lab for Intelligent Networks and Network Security, Xi'an, 710049, China
[5] Shanghai Vocational School of CAAC, Shanghai 200232, P.R. China
zhengff@mail.xjtu.edu.cn

Abstract. This paper considers one parallel machine scheduling problem in which the processing time of a job is a simple linear increasing function of its starting time. The objective is to minimize the makespan, and our focus is on the case with an arbitrary number of parallel machines. We prove that LIST rule is $(1+b_{max})^{\frac{m-1}{m}}$-approximation where m is the number of machines and b_{max} is the maximum deteriorating rate of job. We then propose one heuristic LDR (Largest deteriorating Rate first). The heuristic is proved $(1+b_{min})^{\frac{m-1}{m}}$-approximation where b_{min} is the minimum deteriorating rate. We further show that this ratio is tight when $m = 2, 3$ and 4.

Keywords: Scheduling, Parallel machine, Simple linear deterioration, Makespan, Approximation.

1 Introduction

In classical scheduling theory, job processing times are assumed to be fixed, i.e., independent of their starting times [1]. In real-life applications, however, we often encounter the situations where the processing times are time-dependent. Examples can be found in steel production, machine maintenance and fire fighting, in which any delay in dealing with a task may increase its processing time [2]. Such problems are generally known as scheduling with deterioration effect.

Gupta and Gupta [3], as well as Browne and Yechiali [4], first introduced job's deterioration. They defined *linear deteriorating* job whose actual processing time increases in its starting time. More precisely, the actual processing time of job J_j is defined as $p_j = a_j + b_j t$ ($b_j > 0$), where a_j is the basic processing time, b_j the deteriorating rate, and t the starting time. They showed that sequencing jobs in non-decreasing order of a_j/b_j minimizes the makespan. Mosheiov [5] studied a special case where there is a common basic processing time, i.e., $a_j = a$. For the objective to minimize total flow time or total completion time, it was showed that an optimal schedule is V-shaped with respect to b_j. Mosheiov [6] further introduced the concept of *simple linear deteriorating* job such that $p_j = b_j t$ for each job J_j. Several polynomial algorithms were proposed for the objectives to minimize makespan, flow time, total weighted flow time, maximum lateness, maximum tardiness and the number of tardy jobs, respectively. Cheng and Ding [7] considered a variant with step-deteriorating and gave a pseudo-polynomial algorithm for makespan minimization problem. The above literature focuses on linear deteriorating job scheduling on a single machine, and it is already well done for most scenarios.

For simple linear deteriorating jobs scheduling in the environment of m parallel-machines, the problems become more complex. Mosheiov [8] proved that the makespan minimization problem is strong NP-hard even for two-machine case. An asymptotically optimal heuristic was given in the paper. Chen [9,10] showed that the problem to minimize total completion time is NP-hard even with a fixed number of machines. For the case with two machines, Chen presented an approximation scheme with a parameter dependent worst-case ratio. Ji and Cheng [11] further gave a fully polynomial-time approximation scheme (FPTAS) for the case with any a fixed number of machines. Ji and Cheng [12] investigated three problems to minimize the makespan, total machine load, and total completion time. They proved that all the problem are strongly NP-hard with an arbitrary number of machines and NP-hard in the ordinary sense with a fixed number of machines, respectively. For the former two problems, they proved the non-existence of polynomial time approximation algorithm with a constant ratio when the number of machines is arbitrary, and proposed FPTASs when the number of machines is fixed. Kang and Ng [13] considered the makespan minimization problem with linear deteriorating function (i.e., $p_j = a_j + b_j t$), and also proposed an FPTAS. For more results on linear deteriorating job scheduling, please refer to [14,15,16,17].

We already know by Ji and Cheng [12] that for the m parallel machine scheduling problem to minimize the makespan, when the value of m is arbitrary, there are no polynomial algorithms with constant approximation ratios. In this paper, we present two heuristics for the concerned problem, and investigate their approximation ratios related to parameter m. The remainder of this paper is organized as follows. Section 2 gives problem statement and some notations. Section 3 shows the worst case performance of LIST rule. In Section 4, we present heuristic LDR and its worst case analysis as well.

2 Problem Statement and Notations

There are a set of simple linear deteriorating jobs $\mathcal{I} = \{J_1, J_2, ..., J_n\}$ to be scheduled on m (> 0) identical parallel machines. Each machine processes at most one job at a time and preemption is not allowed. For each job J_j, its actual processing time $p_j = b_j t$, where b_j (> 0) and t are its deteriorating rate and starting time respectively. All the jobs are simultaneously available at time t_0. We assume that $t_0 > 0$ since otherwise if $t_0 = 0$, it is trivial that the makespan is equal to 0 due to $p_j = 0$ ($j \in \{1, ..., n\}$). The objective is to minimize the makespan, i.e., the completion time of the last job. The problem is denoted by $Pm|r_j \geq r_0, p_j = b_j t|C_{max}$.

Denote by M_i ($i \in \{1, ..., m\}$) the ith machine in the system. Given a schedule, let n_i ($i \in \{1, ..., m\}$) be the number of jobs scheduled on machine M_i. Then $n_i = 1, \ldots, n$ for $i \in \{1, ..., m\}$ and $\sum n_i = n$. Let $J_{[i,j]}$ ($j \in \{1, ..., n_i\}$) be the jth job on M_i, and $C_{[i,j]}$ the completion time of the job. By $p_j = b_j t$,

$$\begin{aligned} &C_{[i,1]} = t_0 + p_{[i,1]} = t_0(1 + b_{[i,1]}) \\ &C_{[i,2]} = C_{[i,1]} + p_{[i,2]} = t_0(1 + b_{[i,1]})(1 + b_{[i,2]}) \\ &C_{[i,3]} = t_0 \prod_{l=1}^{3}(1 + b_{[i,l]}) \\ &\ldots \end{aligned}$$

It can be verified that for $i \in \{1, ..., m\}$ and $j \in \{1, ..., n_i\}$,

$$C_{[i,j]} = t_0 \prod_{l=1}^{j}(1 + b_{[i,l]}). \tag{1}$$

Thus, $C_{max} = \max\limits_{i=1,\ldots,m; j=1,\ldots,n_i} C_{[i,j]}$. Since $t_0 > 0$ is an extraneously given constant, we assume without loss of generality that $t_0 = 1$.

3 Heuristic LIST

LIST is a well known heuristic in scheduling theory. It only considers the current loads of processing on machines when to assign a job, and always selects the machine with smallest load. In the following, we present the heuristic and give its worst case analysis.

LIST works as follows:
Assign jobs $\{J_1, J_2, ..., J_n\}$ in sequence, and assign each job to the machine with smallest current load. Ties are broken by selecting the machine with smallest index.

By LIST, J_1 is the first assigned job and J_n is the last one. Let C^T_{max} be the makespan of LIST schedule, and C^*_{max} the makespan of a (possibly unknown) optimal schedule, respectively. We upper bound the ratio C^T_{max}/C^*_{max} in the following. Let $b_{max} = \max\limits_{j=1,\ldots,n} b_j$.

Theorem 1. *For $Pm|r_j \geq r_0, p_j = b_j t|C_{max}$, LIST is $(1 + b_{max})^{\frac{m-1}{m}}$ -approximation.*

Proof. Given jobs $\{J_1, \ldots, J_n\}$. By Formula (1), the starting time of the last job in LIST schedule is $C^T_{max}/(1+b_n)$. Since all the machines are kept busy during $[1, \frac{C^T_{max}}{1+b_n})$,

$$\frac{C^T_{max}}{1+b_n} \leq \left(\prod_{l=1}^{n-1}(1+b_l)\right)^{\frac{1}{m}}.$$

The right-hand side is an upper bound on the starting time of J_n. The inequality is verified when scheduling the first $n-1$ jobs results in the same processing load on each machine. It follows that

$$\begin{aligned} C^T_{max} &\leq (1+b_n)\left(\prod_{l=1}^{n-1}(1+b_l)\right)^{\frac{1}{m}} \\ &= (1+b_n)^{\frac{m-1}{m}}\left(\prod_{l=1}^{n}(1+b_l)\right)^{\frac{1}{m}}. \end{aligned}$$

Since $C^*_{max} \geq (\prod_{l=1}^{n}(1+b_l))^{\frac{1}{m}}$, we have

$$\frac{C^T_{max}}{C^*_{max}} \leq (1+b_n)^{\frac{m-1}{m}} \leq (1+b_{max})^{\frac{m-1}{m}}.$$

This completes the proof.

Theorem 2. *The ratio $(1+b_{max})^{\frac{m-1}{m}}$ is tight for LIST.*

Proof. It suffices to present a specific job instance in which the ratio $(1+b_{max})^{\frac{m-1}{m}}$ is satisfied.

Given an instance $\mathcal{I} = \{J_1, J_2, ..., J_n\}$ where $n = m(m-1)+1$, $b_1 = \cdots = b_{n-1} = 1$ and $b_n = 2^m - 1$. Then $b_{max} = b_n$ and $(1+b_{max})^{\frac{m-1}{m}} = (1+2^m-1)^{\frac{m-1}{m}} = 2^{m-1}$. By *LIST*, $C^T_{max} = 2^{m-1} \cdot 2^m = 2^{2m-1}$. In an optimal schedule, J_n is assigned to one machine and all the previous $n-1$ jobs are equally assigned to the other $m-1$ machines. This yields to $C^*_{max} = 2^m$. Thus,

$$\frac{C^T_{max}}{C^*_{max}} = 2^{m-1}.$$

The theorem is proved.

4 Heuristic LDR

By Formula (1), the load of each machine depends on the set of jobs assigned to the machine but not the processing order of the jobs. Moreover, by the proof of Theorem 2, we observe that the makespan is much related to the deteriorating rate of the last assigned job. Hence, the idea of heuristic LDR (Largest Deteriorating Rate first) is to process jobs in non-increasing order of deteriorating rate. This ensures that the last assigned job is of smallest deteriorating rate and all the machines have balanced loads.

LDR rule:
Assign jobs $\{J_1, J_2, ..., J_n\}$ in non-decreasing order of deteriorating rate, and assign each job to the machine with smallest current load. Ties are broken by selecting the machine with smallest index.

In the following we upper bound the ratio C^R_{max}/C^*_{max} where C^R_{max} is the makespan of LDR schedule and C^*_{max} is the makespan of an optimal schedule. Let $b_{min} = \min_{j=1,...,n} b_j$. Without loss of generality, assume that the n jobs $\{J_1, J_2, ..., J_n\}$ are indexed in non-decreasing order of deteriorating rate. Let $b_{min} = \min\limits_{j=1,...,n} b_j = b_n$.

Theorem 3. *For $Pm|r_j \geq r_0, p_j = b_j t|C_{max}$, LDR is $(1 + b_{min})^{\frac{m-1}{m}}$-approximation.*

The proof of the above theorem is similar to that of Theorem 1, and the only difference lies in that $b_n \leq b_{max}$ for LIST while $b_n = b_{min}$ for LDR. We omit the details of the proof here.

Theorem 4. *When $m = 2, 3$ and 4, the ratio $(1 + b_{min})^{\frac{m-1}{m}}$ is tight for LDR.*

Proof. For the three cases of m, it is sufficient to present specific job instances in which $C^R_{max}/C^*_{max} \geq (1 + b_{min})^{\frac{m-1}{m}}$.

Case 1. $m = 2$.

Given a job instance $\mathcal{I} = \{J_1, J_2, J_3, J_4, J_5\}$ where $b_1 = b_2 = 2^3 - 1$ and $b_3 = b_4 = b_5 = 2^2 - 1$. Since $b_{min} = 2^2 - 1$ and $m = 2$, we have $(1 + b_{min})^{\frac{m-1}{m}} = (1 + 2^2 - 1)^{\frac{1}{2}} = 2$. *LDR* assigns J_1, J_3 and J_5 to machine M_1, and assigns J_2 and J_4 to M_2. $C^R_{max} = 2^3 \cdot 2^2 \cdot 2^2 = 2^7$. In an optimal schedule, J_1 and J_2 are assigned to one machine and the last three jobs are assigned to the other machine. This yields to $C^*_{max} = 2^6$, and the ratio $C^R_{max}/C^*_{max} = 2$.

Case 2. $m = 3$.

Given a job instance $\mathcal{I} = \{J_1, J_2, J_3, J_4, J_5, J_6, J_7\}$ where $b_1 = 2^{12} - 1$, $b_2 = b_3 = 2^{10} - 1$, $b_4 = b_5 = 2^8 - 1$ and $b_6 = b_7 = 2^6 - 1$. Similarly, $(1 + b_{min})^{\frac{m-1}{m}} = (1 + 2^6 - 1)^{\frac{2}{3}} = 2^4$. *LDR* assigns J_1, J_6 and J_7 to machine M_1, assigns J_2 and J_4 to M_2, and assigns the rest two jobs to M_3. $C^R_{max} = 2^{12} \cdot 2^6 \cdot 2^6 = 2^{24}$. In an optimal schedule, J_1 and J_4 are assigned to M_1, J_2 and J_3 are assigned to M_2, and all the remaining jobs are scheduled on M_3. This yields to $C^*_{max} = 2^{20}$, and thus $\frac{C^R_{max}}{C^*_{max}} = 2^4$.

Case 3. $m = 4$.

Given a job instance $\mathcal{I} = \{J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8, J_9\}$ where $b_1 = b_2 = 2^{21} - 1$, $b_3 = b_4 = 2^{18} - 1$, $b_5 = b_6 = 2^{15} - 1$ and $b_7 = b_8 = b_9 = 2^{12} - 1$. $(1 + b_{min})^{\frac{m-1}{m}} = (1 + 2^{12} - 1)^{\frac{3}{4}} = 2^9$. According to *LDR* heuristic, $C^R_{max} =$

$2^{21} \cdot 2^{12} \cdot 2^{12} = 2^{45}$. In an optimal schedule, J_1 and J_5 are assigned to machine M_1, J_2 and J_6 are assigned to M_2, J_3 and J_4 are scheduled on M_3 and the last three jobs are assigned to M_4. $C^*_{max} = 2^{36}$, and $\frac{C^R_{max}}{C^*_{max}} = 2^9$.

This completes the proof.

Acknowledgements. This work was partially supported by NSF of China under Grants 71071123, 70832005, 60736027, 60921003 and 71090404 / 71090400.

References

1. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems, 2nd edn. Prentice-Hall, Englewood Cliffs (2000)
2. Cheng, T., Ding, Q., Lin, B.: A concise survey of scheduling with time-dependent processing times. European Journal of Operational Research 152, 1–13 (2004)
3. Gupta, J., Gupta, S.: Single facility scheduling with nonlinear processing times. Computers and Industrial Engineering 14, 387–393 (1988)
4. Browne, S., Yechiali, U.: Scheduling deteriorating jobs on a single processor. Operations Research 38, 495–498 (1990)
5. Mosheiov, G.: V-shaped policies for scheduling deteriorating jobs. Operations Research 39, 979–991 (1991)
6. Mosheiov, G.: Scheduling jobs under simple linear deterioration. Computers and Operations Research 21, 653–659 (1994)
7. Cheng, T., Ding, Q.: Single machine scheduling with step-deteriorating processing times. European Journal of Operational Research 134, 623–630 (2001)
8. Mosheiov, G.: Multi-machine scheduling with linear deterioration. INFOR 36(4), 205–214 (1998)
9. Chen, Z.L.: Parallel machine scheduling with time dependent processing times. Discrete Applied Mathematics 70, 81–93 (1996)
10. Chen, Z.L.: Erratum to Parallel machine scheduling with time dependent processing times. Discrete Applied Mathematics 70, 81–93 (1996); Discrete Applied Mathematics 75, 103 (1997)
11. Ji, M., Cheng, T.C.E.: Parallel-machine scheduling with simple linear deterioration to minimize total completion time. European Journal of Operational Research 188, 342–347 (2008)
12. Ji, M., Cheng, T.C.E.: Parallel-machine scheduling of simple linear deteriorating jobs. Theoretical Computer Science 410, 38–40 (2009)
13. Kang, L., Ng, C.T.: A note on a fully polynomial-time approximation scheme for parallel-machine scheduling with deteriorating jobs. International Journal of Production Economics 109, 180–184 (2007)
14. Alidaee, B., Womer, N.: Scheduling with time dependent processing times: Review and extentions. Journal of the Operational Research Society 50, 711–720 (1999)
15. Cheng, T., Kang, L., Ng, C.: Due-date assignment and single machine scheduling with deteriorating jobs. Journal of Operational Research Society 55, 198–203 (2004)
16. Lodree, E., Gerger, C.: A note on the optimal sequence position for a rate-modifying activity under simple linear deterioration. European Journal of Operational Research 201, 644–648 (2010)
17. Cheng, Y., Sun, S.: Scheduling linear deterorating jobs with rejection on a single machine. European Journal of Operational Research 194, 18–27 (2009)

A Comprehensive Study of an Online Packet Scheduling Algorithm

Fei Li[⋆]

Department of Computer Science, George Mason University,
Fairfax, VA 22030, USA
http://www.cs.gmu.edu/~lifei

Abstract. We study the *bounded-delay model* for Qualify-of-Service buffer management. Time is discrete. There is a buffer. Unit-length jobs (also called *packets*) arrive at the buffer over time. Each packet has an integer release time, an integer deadline, and a positive real value. A packet's characteristics are not known to an online algorithm until the packet actually arrives. In each time step, at most one packet can be sent out of the buffer. The objective is to maximize the total value of the packets sent by their respective deadlines in an online manner. An online algorithm's performance is usually measured in terms of *competitive ratio*, when this online algorithm is compared with a clairvoyant algorithm achieving the best total value. In this paper, we study a simple and intuitive online algorithm. We analyze its performance in terms of competitive ratio for the general model and a few important variants.

Keywords: online algorithm, competitive analysis, buffer management, packet scheduling.

1 Model Description

We consider the *bounded-delay model* introduced in [6][7]. Time is discrete. The t-th *(time) step* presents the time interval $(t-1,\ t]$. There is a buffer and unit-length jobs (also called *packets*) arrive at the buffer over time. Each packet p has an integer release time $r_p \in \mathbb{Z}^+$, an integer deadline $d_p \in \mathbb{Z}^+$, and a positive real value $v_p \in \mathbb{R}^+$. A packet p's characteristics are not known to an online algorithm until p actually arrives at the buffer at time r_p. In each step, at most one packet in the buffer can be sent. A packet p is said to be *successfully sent* at time t if $r_p \leq t \leq d_p$. The objective is to maximize the total value of the packets that are successfully sent in an online manner.

As people have noted, the offline version of this problem can be solved efficiently using the Hungarian algorithm [8] in time $O(n^3)$, where n is the number of packets in the input instance.

In the framework of *competitive analysis* which provides worst-case guarantees, an online algorithm's performance is measured in terms of *competitive ratio* [1]. For a maximization problem, an online algorithm is called *c-competitive*

[⋆] Research partially supported by NSF Grant CCF-0915681.

if for *any* finite instance, its total value is no less than $1/c$ times of what an optimal offline algorithm achieves. In competitive analysis, an input instance is allowed to be generated in an adversarial way so as to maximize the competitive ratio. The upper bound of competitive ratio is achieved by some online algorithms. A competitive ratio strictly less than the lower bound cannot be reached by any online algorithm. If an online algorithm has its competitive ratio same as the lower bound, we say that this online algorithm is *optimal*. For the bounded-delay model, the currently best known result is $2\sqrt{2}-1 \approx 1.828$ [5] and the lower bound is $(1+\sqrt{5})/2 \approx 1.618$ [6][3]. If an online algorithm decides which packet to send only based on the contents of its current buffer, and independent of the packets that have already been released and processed, we call it *memoryless*.

In this paper, we study a simple, intuitive memoryless online algorithm called MG ('Modified Greedy'). We analyze MG's performance in terms of competitive ratio for the general bounded-delay model and some important variants. Define a packet p's *slack-time* s_p as the difference between its deadline d_p and release time r_p, $s_p = d_p - r_p$. The variants that we consider include:

– *Agreeable deadline* setting. In an agreeable deadline instance, for any two packets p and q with $r_p \leq r_q$, we have $d_p \leq d_q$. This variant has been studied in [9].
– *Anti-agreeable deadline* setting. In an anti-agreeable deadline instance, for any two packets p and q with $r_p \leq r_q$, we have $d_p \geq d_q$.
– *Agreeable value* setting. In an agreeable value instance, for any two packets p and q with $r_p \leq r_q$, we have $v_p \leq v_q$.
– *Anti-agreeable value* setting. In an anti-agreeable value instance, for any two packets p and q with $r_p \leq r_q$, we have $v_p \geq v_q$.
– *Agreeable deadline/value* setting. In an agreeable deadline/value instance, for any two packets p and q with $d_p \leq d_q$, we have $v_p \leq v_q$.
– *Anti-agreeable deadline/value* setting. In an anti-agreeable deadline/value instance, for any two packets p and q with $d_p \leq d_q$, we have $v_p \geq v_q$.
– *Agreeable slack-time/value* setting. In an agreeable slack-time/value instance, for any two packets p and q with $s_p \leq s_q$, we have $v_p \leq v_q$.
– *Anti-agreeable slack-time/value* setting. In an anti-agreeable slack-time/value instance, for any two packets p and q with $s_p \leq s_q$, we have $v_p \geq v_q$.

Our results are summarized in Table 1. Note that the lower bounds shown in Table 1 are the lower bounds of MG's performance but not the lower bounds for any online algorithms.

In the following, we present the online algorithm MG in Section 2 and analyze its performance in Section 3.

2 Algorithm MG

The idea of designing MG is motivated by the *greedy algorithm*: In each step, the highest-value pending packet is sent. This algorithm is proved 2-competitive [6][7]. In one attempt to beat the greedy algorithm in competitiveness, Chin et al. [2] proposed an algorithm called EDF_α, bearing the idea of sending the earliest-deadline

Table 1. Summary of MG's performance for the bounded-delay model and its variants. The results without references are the work presented in this paper. In this table, $\phi = (1+\sqrt{5})/2 \approx 1.618$.

models	upper bounds	lower bounds	notes
general	2	2 [10]	A detailed analysis of the lower bound is given in this paper.
agreeable deadline	ϕ [9]	ϕ [3]	MG is optimal.
anti-agreeable deadline	2	2 [10]	-
agreeable value	2	2 [10]	-
anti-agreeable value	1	1	MG is optimal.
agreeable deadline/value	ϕ	ϕ [3]	MG is optimal.
anti-agreeable deadline/value	1	1	MG is optimal.
agreeable slack-time/value	ϕ	1	-
anti-agreeable slack-time/value	1	1	MG is optimal.

packet with a sufficiently large value (for instance, at least $1/\alpha$ times of the highest value of a pending packet where $\alpha \geq 1$). Note that EDF_α generalizes the greedy algorithm, which is EDF_1. Same as the greedy algorithm, EDF_α is asymptotically not better than 2-competitive. For EDF_α, it is possible that the expiring packet in the algorithm's buffer is the one that an optimal offline algorithm sends and this packet has only a slightly less value than the packet that EDF_α sends.

Recall that a memoryless online algorithm makes its decision only based on the contents of its current buffer. Thus, it is natural to send a packet from a set of packets, all of which are eligible of being sent successfully under the assumption of no future arrivals. We consider *provisional schedules*. A provisional schedule [4][5] at time t is a schedule specifying the set of pending packets to be transmitted and for each it specifies the delivery time, assuming no newly arriving packets. An optimal provisional schedule achieves the maximum total value among all the provisional schedules. At the beginning of each step, we calculate an optimal provisional schedule S and the packets in S are arranged in a *canonical* order: increasing order of deadlines, with ties broken in decreasing order of values.

Let e denote the first packet in S and h denote the first highest-value packet in S. Motivated by the idea of EDF_α, we would like to send a packet with a sufficiently large value compared with v_h. At the same time, from the tight example for EDF_α, we would like to send a packet to compensate the potential loss due to not sending the earliest-deadline packet e. Thus, we send a packet f in the optimal provisional schedule satisfying $v_f \geq v_h/\alpha$ if $f = e$ and $v_f \geq \max\{\beta v_e,\ v_h/\alpha\}$ if $f \neq e$, where $\alpha,\ \beta \geq 1$. In order to guarantee that at least one packet in S can be a candidate packet for f, we have to have $\alpha \geq \beta$ since if $v_e < v_h/\alpha$, we should have $v_h \geq v_f \geq \max\{\beta v_e,\ \alpha v_e\} \geq \max\{\beta,\ \alpha\} v_e$. The algorithm MG is described in Algorithm 1.

Note that MG generalizes EDF_α (and the greedy algorithm). If $\alpha = 1$ (hence $\beta = 1$ since $\alpha \geq \beta \geq 1$), MG is the greedy algorithm. If $\beta = 1$, MG is no-worse than EDF_α in competitiveness.

Algorithm 1. MG (t, $1 \leq \beta \leq \alpha$)

```
1. Calculate an optimal provisional schedule $S$. All the packets in $S$ are sorted in a
   canonical order: increasing order of deadlines, with ties broken in decreasing order
   of values. In $S$, let $e$ denote the first packet; let $h$ denote the first highest-value
   packet.
2. if $v_e \geq v_h/\alpha$ then
3.    send $e$;
4. else
5.    send the first packet $f$ satisfying $v_f \geq \max\{v_h/\alpha, \beta v_e\}$.
6. end if
```

Theorem 1. *If $\beta = 1$, MG is no-worse than EDF$_\alpha$ in competitiveness.*

Proof. We inductively prove that (1) MG with $\beta = 1$ and EDF$_\alpha$ share the same buffer at any time; (However, we note here that MG's optimal provisional schedule may not be identical to EDF$_\alpha$'s buffer.) and (2) in each step, the charged value to MG is no less than the charged value to EDF$_\alpha$.

Assume MG sends $f \neq e$. EDF$_\alpha$ must send f as well since all the packets with values $\geq v_h/\alpha$ must be in MG's optimal provisional schedule. Assume MG sends the e-packet and EDF$_\alpha$ sends a packet p not in MG's optimal provisional schedule. If EDF$_\alpha$ does not send e in its schedule, we have $v_e \geq v_p$ and we can use e to replace p for EDF$_\alpha$. □

3 Analysis

Let OPT denote an optimal offline algorithm and $\mathcal{O}$ denote the set of packets that OPT sends. Let ADV denote a (modified) adversary. In our proof, we will create ADV and make sure that ADV gains a total value no less than $\sum_{p \in \mathcal{O}} v_p$.

3.1 The General Setting

Theorem 2. *MG is 2-competitive for the bounded-delay model, for any $1 \leq \beta \leq \alpha \leq 2$.*

Proof. We assume that there exists an adversary called ADV. We modify ADV such that ADV and MG share the same buffer at the beginning of each step. ADV does not have to send every packet in its buffer. In a step, MG sends the packet f.

1. Assume ADV sends the same packet f in this step.
   ADV and MG gain the same value.
2. Assume ADV sends a packet j ($\neq f$) with $d_j < d_f$.
   We modify ADV by sending both j and f in the current step. We then insert j into ADV's buffer as a gift packet. As assumed, j is in MG's buffer at the beginning of this step. From the canonical order and MG choosing f but not j to send, we have $v_j \leq v_f$. Then $v_j + v_f \leq 2v_f$.

3. Assume ADV sends a packet j ($\neq f$) with $d_j > d_f$.
   As assumed, j is in MG's buffer at the beginning of this step. No matter $f = e$ or $f \neq e$, we have $v_f \geq v_h/\alpha \geq v_j/\alpha \geq v_j/2$. Note that $v_f < v_j$ (and $d_f < d_j$) since otherwise, ADV prefers to sending f instead of j. We then insert j into ADV's buffer to replace f.

At the end of this step, ADV and MG share the same buffer again. The modifications that we make favor the adversary but not MG. In this step, ADV's modified gain is bounded by 2 times of what MG achieves. □

Theorem 3. *MG is asymptotically no better than* 2*-competitive for the bounded-delay model, with* $\alpha = \beta = \phi$*.*

A sketched proof of Theorem 3 has been given in [10]. We detail the analysis in this paper.

Proof. We construct an example to prove Theorem 3. We use ∞ in the deadline field of a packet to show that this packet's deadline is very large. Let $n = 2^k$. The packets are released in a stage-manner. There are $\log n = k$ stages. The superscript of a packet shows the stage in which it is released.

At the beginning of step 1, there are 3 packets in MG's buffer. The adversary has the same buffer. These 3 packets are $e_1^1 := (1+\epsilon,\ 2)$, $f_1^1 := (\phi-\epsilon,\ 2^{k+1}-k)$, and $h_1^1 := (\phi,\ \infty)$. MG sends h_1^1, and e_1^1 is dropped out of the buffer due to its deadline.

In each of the following (2^k-k+1) time steps, say step i, a group of 3 packets are released: $e_i^1 := (1+\epsilon,\ i+1)$, $f_i^1 := (\phi-\epsilon,\ 2^{k+1}-k)$, and $h_i^1 := (\phi,\ \infty)$. In step i, MG sends h_i^1 and drops e_i^1 due to its deadline. At the end of the (2^k-k+1)-th step, MG's buffer is full of (2^k-k+1) f_i^1-packets ($\forall i = 1,\ 2,\ \ldots,\ 2^k-k+1$). The first stage ends. The length of stage 1 guarantees that no f_i^1 packet, especially packet f_1^1, becomes the first packet in the buffer.

At the beginning of step 2^k-k+1, the second stage starts. The adversary releases a pair of packets $f_1^2 := (\phi(\phi-\epsilon)-\epsilon,\ 2^{k+1}-k+1)$ and $h_1^2 := (\phi^2,\ \infty)$. The newly released packets have later deadlines and are sorted canonically after the packets already in MG's buffer. MG sends h_i^2. Stage 2 contains $2^{k-1}-k+2$ steps. The length of stage 2 guarantees that no packet f_i^2 becomes the first packet in the buffer. In each of those $2^{k-1}-k+2$ steps, say step i, 2 packets are released $f_i^2 := (\phi(\phi-\epsilon)-\epsilon,\ 2^{k+1}-k+1)$ and $h_i^2 := (\phi^2,\ \infty)$. MG sends h_i^2 in step i. Stage 2 is half as long as stage 1.

We repeat this pattern in each stage, for k stages. Stage $i+1$ is half as long as stage i. In each step j of stage i, 2 packets are released, $f_j^i := (\phi(w_{f_1^{i-1}} - \epsilon),\ 2^{k+1}-k+i)$ and $h_j^i := (\phi^i,\ \infty)$. MG sends h_j^i in step j. In the last stage, which is step 2^{k+1}, the adversary only releases 2 packets $f_1^k := (\phi^k,\ 2n)$ and $h_1^k := (\phi^{k+1}+\epsilon,\ \infty)$. MG sends h_1^k and f_1^k is dropped out of the buffer due to its deadline.

For each step in stage i, MG only delivers the h^i packets, and eventually, all packet f^i are dropped out of the buffer due to their deadlines. On the contrary, the adversary sends all f^i packets and all h^i packets. A routine calculation shows that the optimal weighted throughput is nearly twice MG's weighted throughput. We remove ϵ in the following calculation for the sake of clearness.

$$\begin{aligned} c &= \frac{2\left(\phi^0 \cdot 2^k + \phi^1 \cdot 2^{k-1} + \ldots + \phi^k \cdot 2^0\right) + \phi^{k+1}}{\left(\phi^0 \cdot 2^k + \phi^1 \cdot 2^{k-1} + \ldots + \phi^k \cdot 2^0\right) + \phi^{k+1}} \\ &= \frac{2\left(\phi^0 \cdot 2^k\right)\left(\frac{\phi^0}{2^0} + \frac{\phi^1}{2^1} + \frac{\phi^2}{2^2} + \ldots + \frac{\phi^k}{2^k}\right) + \phi^{k+1}}{\left(\phi^0 \cdot 2^k\right)\left(\frac{\phi^0}{2^0} + \frac{\phi^1}{2^1} + \frac{\phi^2}{2^2} + \ldots + \frac{\phi^k}{2^k}\right) + \phi^{k+1}} = \frac{2^{k+1}\frac{1-\left(\frac{\phi}{2}\right)^{k+1}}{1-\frac{\phi}{2}} + \phi^{k+1}}{2^k\frac{1-\left(\frac{\phi}{2}\right)^{k+1}}{1-\frac{\phi}{2}} + \phi^{k+1}} \\ &= \frac{2^{k+1} - \phi^{k+1} + \phi^{k+1} - \frac{\phi^{k+2}}{2}}{2^k - \frac{\phi^{k+1}}{2} + \phi^{k+1} - \frac{\phi^{k+2}}{2}} = \frac{2\left(\frac{2}{\phi}\right)^k - \frac{\phi^2}{2}}{\left(\frac{2}{\phi}\right)^k - \frac{1}{2}} = 2. \end{aligned}$$

□

3.2 The Agreeable Deadline Setting

In [9], the authors have shown that MG is ϕ-competitive for agreeable deadline instances. The lower bound ϕ constructed in [3] for the general model holds as well for scheduling packets with agreeable deadlines and MG. We list MG's performance in the agreeable deadline setting here for its optimality and significance. We include this variant for comparison with others.

3.3 The Anti-agreeable Deadline Setting

Both Theorem 2 and Theorem 3 hold for anti-agreeable deadline instances. Both the upper bound and lower bound for MG are 2.

3.4 The Agreeable Value Setting

Both Theorem 2 and Theorem 3 hold for anti-agreeable deadline instances. Both the upper bound and lower bound for MG are 2.

3.5 The Anti-agreeable Value Setting

Theorem 4. *MG is* 1*-competitive for the anti-agreeable value setting when* $\alpha = \infty$*. MG is optimal.*

Proof. When $\alpha = \infty$, MG sends the earliest-deadline packet e in the optimal provisional schedule in each step. To prove Theorem 4, we only need to inductively show that for each step, an optimal offline algorithm OPT sends e in each step as well. In anti-agreeable value instances, any later released packet has a value $\leq v_e$. If any later released packet belongs to $\mathcal{O}$, so does e. If no later released packet belongs to $\mathcal{O}$, OPT sends e to maximize its total gain. Thus, OPT sends e in each step. □

3.6 The Agreeable Deadline/Value Setting

The lower bound ϕ constructed in [3] for the general model holds as well for agreeable deadline/value instances.

Theorem 5. *MG is ϕ-competitive for the agreeable deadline/value setting when $\alpha = \beta = \phi^2 \approx 2.618$. MG is optimal.*

Proof. We are using a charging scheme to prove Theorem 5. Let OPT denote an optimal offline algorithm. Without loss of generality, we assume that OPT only accepts $\mathcal{O}$-packets and sends them in EDF manner. Let Q^{OPT} denote OPT's buffer.

At time t, let the optimal provisional schedule be S and we index the buffer slots as $t,\ t+1,\ \ldots$. The packets in S are sorted in increasing deadline order, with ties broken in decreasing value order and these packets are buffered in slots $t,\ t+1,\ \ldots,\ t+|S|-1$ consecutively. The packets not in S are appended at the end of S. Let us study the optimal provisional schedule S at first. The packets in S thus are grouped into multiple (≥ 1) *batches of packets* $G_1,\ G_2,\ \ldots$, in order of *strictly increasing deadlines.* The packets in the same batch share the same deadline. (Note that G_1 is the first batch in S.) We have

Remark 1. All the packets in the same batch share the same deadline. For any two batches G_i and G_j with indexes $i < j$, all the packets in G_i have strictly earlier deadlines and strictly lower values than all the packets in G_j.

We will introduce a charging scheme and this charging scheme may use the following observations.

Remark 2. In the agreeable deadline/value setting, if a packet p is inserted into the optimal provisional schedule, then all the packets with value $> v_p$ are shifted into one buffer slot later since they have strictly larger deadlines. Also, for any two time steps, the relative order among the packets in both MG's optimal provisional schedules is not changed.

Lemma 1. *In the agreeable deadline/value setting, if a packet p is evicted out of MG's optimal provisional schedule at time t, then in each step from time t till p's deadline d_p, MG's optimal provisional schedules for these steps do not contain any packet with a value $< d_p$.*

Proof. If a packet p is evicted out of MG's optimal provisional schedule at time t, then either $d_p < t$ or in each of the buffer slots $t,\ t+1,\ \ldots,\ d_p$, MG's current optimal provisional schedule at time t buffers one packet with value $> v_p$. From Remark 1 and the assumption of agreeable deadline/value, d_p should not be larger than those of packets in the batch G_1.

– Assume MG sends the e-packet in a step before d_p.
  Then for those packets arranged in the buffer slots belonging to batch G_1, they have their deadlines no smaller than d_p and they are *tight*, that is, they cannot be shifted into later buffer slots and provide buffer slots to

accommodate less-value packets with no-later deadlines (see Remark 2). For packets in batches G_2, G_3, …, if any, they have strictly larger deadlines than d_p and strictly larger values than v_p.

– Assume MG sends a packet $f \neq e$ in a step before d_p.
All the unsent packets in the optimal provisional schedule can be shifted by at most one step to their later steps and the relative order among all these packets keep unchanged (see Remark 1 and Remark 2). Any newly released packets with later deadlines have no smaller values. Any newly released packets with values $< v_p$ are rejected by MG's optimal provisional schedules since all the packets with deadlines $= d_p$ are tight. Thus, for the new optimal provisional schedule generated at the beginning of the next step, Lemma 1 still holds. □

Lemma 2. *Consider a* chain of k steps*. In the steps* $1, 2, \ldots, k$ *(these steps may not be continues), we charge OPT the values* $v_{q_1}, v_{q_2}, \ldots, v_{q_k}$ *and MG the values* $v_{p_1}, v_{p_2}, \ldots, v_{p_k}$*, respectively. If for all* i *with* $1 \leq i \leq k-1$*, we have* $v_{q_i} \leq \alpha \cdot v_{p_i}$*, and if* $v_{q_i} \leq v_{p_{i+1}}$ *and* $v_{q_k} \leq v_{p_k}$*, then* $\sum_{i=1}^{k} v_{q_i} \leq \frac{1}{\alpha^k - 1}\left(\left(2 - \frac{1}{\alpha}\right)\alpha^k - \alpha\right)\sum_{i=1}^{k} v_{p_i}$.

Proof.

$$\begin{aligned}
\frac{\sum_{i=1}^{k} v_{q_i}}{\sum_{i=1}^{k} v_{p_i}} &= \frac{v_{q_1} + v_{q_2} + \cdots + v_{q_k}}{v_{p_1} + v_{p_2} + \cdots + v_{p_k}} \leq \frac{v_{q_1} + v_{q_2} + \cdots + v_{q_k}}{\frac{v_{q_1}}{\alpha} + \max\{v_{q_1}, \frac{v_{q_2}}{\alpha}\} + \cdots + v_{p_k}} \\
&\leq \frac{\frac{v_{q_2}}{\alpha} + v_{q_2} + \cdots + v_{q_k}}{\frac{v_{q_2}}{\alpha^2} + \frac{v_{q_2}}{\alpha} + \cdots + v_{p_k}} \leq \cdots \leq \frac{\frac{v_{q_{k-1}}}{\alpha^{k-2}} + \cdots + \frac{v_{q_{k-1}}}{\alpha} + v_{q_{k-1}} + v_{q_k}}{\frac{v_{q_{k-1}}}{\alpha^{k-1}} + \cdots + \frac{v_{q_{k-1}}}{\alpha^2} + v_{p_{k-1}} + v_{p_k}} \\
&\leq \frac{\frac{v_{q_{k-1}}}{\alpha^{k-2}} + \cdots + \frac{v_{q_{k-1}}}{\alpha} + v_{q_{k-1}} + v_{q_k}}{\frac{v_{q_{k-1}}}{\alpha^{k-1}} + \cdots + \frac{v_{q_{k-1}}}{\alpha^2} + \frac{v_{q_{k-1}}}{\alpha} + \max\{v_{q_k}, v_{q_{k-1}}\}} \\
&\leq \frac{\frac{v_{q_{k-1}}}{\alpha^{k-2}} + \cdots + \frac{v_{q_{k-1}}}{\alpha} + v_{q_{k-1}} + v_{q_{k-1}}}{\frac{v_{q_{k-1}}}{\alpha^{k-1}} + \cdots + \frac{v_{q_{k-1}}}{\alpha^2} + \frac{v_{q_{k-1}}}{\alpha} + v_{q_{k-1}}} \\
&= \frac{\frac{1-\alpha^{1-k}}{1-\alpha^{-1}} + 1}{\frac{1-\alpha^{-k}}{1-\alpha^{-1}}} = \frac{(2-\alpha^{-1})\alpha^k - \alpha}{\alpha^k - 1}.
\end{aligned}$$

Note that when $\alpha \geq 1$, $\frac{1}{\alpha^k - 1}\left(\left(2 - \frac{1}{\alpha}\right)\alpha^k - \alpha\right) \leq 2 - \frac{1}{\alpha}$. Also, note $\phi + \frac{1}{\phi^2} = 2$, we have

Corollary 1. *Consider a* chain of k steps*. In the steps* $1, 2, \ldots, k$ *(these steps may not be continues), we charge OPT the values* $v_{q_1}, v_{q_2}, \ldots, v_{q_k}$ *and ON the values* $v_{p_1}, v_{p_2}, \ldots, v_{p_k}$*. If for all* i *with* $1 \leq i \leq k-1$*, we have* $v_{q_i} \leq \alpha \cdot v_{p_i}$*, and if* $v_{q_i} \leq v_{p_{i+1}}$*, and* $v_{q_k} \leq v_{p_k}$*, then we have* $\sum_{i=1}^{k} v_{q_i} \leq \phi \sum_{i=1}^{k} v_{p_i}$ *when* $\alpha = \phi^2$*.*

We say that a chain of steps is *open* if we have not charged the values to OPT and MG in these steps. Otherwise, we say that it is *closed*.

Definition 1 (Canonical Order). *Packets in MG's optimal provisional schedule are order in a* canonical order*: in increasing order of deadlines, with ties broken in decreasing order of values.*

Our charging scheme guarantees the following three invariants:

I_1. In each step or in a closed chain of a group of steps, the total charged values to OPT are bounded by ϕ times of the total charged values to MG. Chains do not share steps.

I_2. For any packet q in OPT's buffer, if v_q has not been charged to OPT in our charging scheme, then q must map uniquely to a packet p in MG's optimal provisional schedule with $v_q \leq v_p$ and $d_q \leq d_p$. (p may be the packet q itself.) In the canonical order, for any packet j before p in MG's optimal provisional schedule S, if p is not in S, then we have $v_j \geq v_q$.

I_3. A packet p in MG's optimal provisional schedule S may correspond to at most one open chain and v_p is no less than the value of the packet OPT sends in the last step of this open chain. If p corresponds to an open chain and is mapped by a packet in OPT's buffer, p is called *overloaded*. If p is overloaded, then any packet before p in S is overloaded as well.

Note that Invariant I_1 results in Theorem 5 automatically.

The charging scheme is described below. We consider packet arrivals and packet deliveries separately.

Packet arrivals. For any packet p evicted out of MG's optimal provisional schedule S due to accepting a new arrival p', we have $v_{p'} \geq v_p$ and $d_{p'} \geq d_p$ in the agreeable deadline/value setting. After dropping p, MG has at least one packet q in S such that q is not mapped by a packet in OPT's buffer, due to Invariant I_2. In the canonical order of S, we pick up the first packet not in mapping and let it be q. q should have a deadline $\geq v_p$ and thus, $v_q \geq v_p$, due to the assumption of agreeable deadline/value setting. Furthermore, any packet in MG's current optimal provisional schedule has a no-less value and no-earlier deadline than p. We transfer the open chain mapping to p, if any, to q. Hence for packet arrivals, all the invariants hold.

Packet deliveries. In each step, OPT sends the earliest-deadline packet q in its buffer. MG sends either e or $f \neq e$. Remember that we use S denotes MG's optimal provisional schedule and the packets in S are sorted in a canonical order.

Assume MG sends e and OPT sends $q \notin S$ or OPT sends $q = e$. From Invariant I_2, if q has not been charged to OPT, then $v_q \leq v_e$. Assume q maps to p in S. $v_q \leq v_p \leq v_e$. We charge OPT v_q and the packets in the open chain mapping to e, if any. We close the open chain. The ratio of total charged values of this chain or this single step is bounded by ϕ (see Corollary 1).

Assume MG sends e and OPT sends $q \in S$ with $q \neq e$. Due to Invariants I_2 and I_3, there is no overloaded packets in MG's optimal provisional schedule. Otherwise, OPT sends a packet with an earlier deadline than d_q and less-value than v_e since it sends packets in the EDF order. We start a new open chain from this step mapping to q in MG's optimal provisional schedule. Note that q is not an overloaded packet yet since it is not mapped by any packet in OPT's buffer.

Assume MG sends $f \neq e$ and OPT sends $q \notin S$ or OPT sends $q = e$. From Algorithm 1, we have $v_f \geq \alpha v_e = \phi^2 v_e$. If q is evicted out of the provisional schedule, we have $v_q \leq v_e$ (from Lemma 1). We close the open chain if e belongs to any one. The ratio of total charged values of this chain or this single step is bounded by ϕ (see Corollary 1).

Assume MG sends $f \neq e$ and OPT sends $q \in S$ with $d_q < d_f$.

– Assume $f = h$.
  We have $v_q < v_h/\alpha = v_f/\alpha = v_f/\phi^2$.
  If $q = e$, we close the open chain mapping to e, if any. We also charge v_h to OPT in this step. The ratio of total charged values of this chain or this single step is bounded by ϕ (see Corollary 1).
  If $q \neq e$, then no open chains exist since otherwise e is a candidate packet for OPT to send. We charge OPT the value $v_q + v_f$ in this step and MG the value v_f. Furthermore, we split this step into two fractional steps: In one fractional step, OPT is charged a value v_f and MG v_f/ϕ. In this single fractional step, the gain ratio is ϕ. In another fractional step, we charge OPT the value v_q and MG the value $v_f/\phi^2 \geq v_q/\phi^2$. This step maps to q in MG's optimal provisional schedule at the end of this step since e with $d_e \geq t$ is not the packet q.
– Assume $f \neq h$.
  If q is not in MG's optimal provisional schedule S, q must map to a packet $p \in S$ and $v_q \leq v_e$. From Algorithm 1, we have $v_f \geq \alpha v_e = \alpha v_q = \phi^2 v_q$. f is not in any open chain (from Invariant I_3). We close the open chain, if any, mapping to p. We also charge v_f to OPT in this step. The ratio of total charged values of this chain or this single step is bounded by ϕ (see Corollary 1).
  If q is in S, then q is not in any open chain, from Invariant 1. We charge OPT the value $v_q + v_f$ in this step and MG the value v_f. Furthermore, we split this step into two fractional steps: In one fractional step, OPT is charged a value v_f and MG v_f/ϕ. In this single fractional step, the gain ratio is ϕ. In another fractional step, we charge OPT the value v_q and MG the value $v_f/\phi^2 \geq v_q/\phi^2$. This step maps to q in MG's optimal provisional schedule at the end of this step since e with $d_e \geq t$ is not the packet q.

Assume MG sends $f \neq e$ and OPT sends $q \in S$ with $d_q > d_f$. Due to Invariants I_2 and I_3, there is no overloaded packets in MG's optimal provisional schedule. From Algorithm 1, we have $v_q > v_f \geq \alpha v_e = \phi^2 v_e$. We start a new open chain from this step mapping to q in MG's optimal provisional schedule. Note that q is not an overloaded packet yet since it maps no packet in OPT's buffer. □

3.7 The Anti-Agreeable Deadline/Value Setting

Consider the anti-agreeable deadline/value setting. In MG's optimal provisional schedule, for any two packets p and q with $d_p < d_q$, we have $v_p \geq v_q$. Applying the same proof of Theorem 4, we have

Theorem 6. *MG is* 1*-competitive for the anti-agreeable deadline/value setting when* $\alpha = \infty$*. MG is optimal.*

3.8 The Agreeable Slack-Time/Value Setting

Lemma 3. *In the agreeable slack-time/value setting, if a packet* p *is evicted out of MG's optimal provisional schedule at time* t*, then from time* t *till* p*'s deadline* d_p*, all the MG's optimal provisional schedules do not contain any packet with a value* $< v_p$*.*

Proof. If a packet p is evicted out of MG's optimal provisional schedule at time t, then either $d_p < t$ or in each of the buffer slots $t, \ t+1, \ \ldots, \ d_p$, MG's current optimal provisional schedule at time t buffers one packet with value $> v_p$.

In each step, MG either sends e or $f \neq e$. For time t when a packet p is rejected, those packets unsent by MG but staying in MG's optimal provisional schedule at time t are tight and cannot be shifted into later buffer slots. Note that for any two packets with the same deadline, the earlier released one has a larger slack time, hence, a larger value. Thus, the later released packet is preferred to be evicted if two packets share the same deadline and MG's optimal provisional schedule cannot accommodate both. Lemma 3 holds. □

Using Lemma 3, we apply the proof of Theorem 5 directly and have

Theorem 7. *MG is* ϕ*-competitive for the agreeable slack-time/value setting when* $\alpha = \beta = \phi = (1 + \sqrt{5})/2 \approx 1.618$*.*

3.9 The Anti-Agreeable Slack-Time/Value Setting

Property 1. Consider the anti-agreeable slack-time/value setting. In MG's optimal provisional schedule, for any two packets p and q with $d_p < d_q$, we have $v_p \geq v_q$.

Property 1 can be proved inductively. Assume at time t, Property 1 holds. Consider a packet p in the optimal provisional schedule at the end of step t. We have $r_p \leq t < d_p$. For any released packet q at time $t+1$, if $d_q < d_p$, we have $s_q = d_q - (t+1) < d_p - t = s_p$ and $v_q > v_p$. Thus, Property 1 holds again. Property 1 results in that all the e-packets in the optimal provisional schedules are $\mathcal{O}$-packet. Applying a slightly modified version of the proof of Theorem 4, we have

Theorem 8. *MG is* 1*-competitive for the anti-agreeable slack-time/value setting when* $\alpha = \infty$*. MG is optimal.*

References

1. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
2. Chin, F.Y.L., Chrobak, M., Fung, S.P.Y., Jawor, W., Sgall, J., Tichy, T.: Online competitive algorithms for maximizing weighted throughput of unit jobs. Journal of Discrete Algorithms 4(2), 255–276 (2006)
3. Chin, F.Y.L., Fung, S.P.Y.: Online scheduling with partial job values: Does time-sharing or randomization help? Algorithmica 37(3), 149–164 (2003)
4. Chrobak, M., Jawor, W., Sgall, J., Tichy, T.: Online scheduling of equal-length jobs: Randomization and restart help? SIAM Journal on Computing (SICOMP) 36(6), 1709–1728 (2007)
5. Englert, M., Westermann, M.: Considering suppressed packets improves buffer management in QoS switches. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 209–218 (2007)
6. Hajek, B.: On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. In: Proceedings of 2001 Conference on Information Sciences and Systems (CISS), pp. 434–438 (2001)
7. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. SIAM Journal on Computing (SICOMP) 33(3), 563–583 (2004)
8. Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly 2, 83–97 (1955)
9. Li, F., Sethuraman, J., Stein, C.: An optimal online algorithm for packet scheduling with agreeable deadlines. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 801–802 (2005)
10. Li, F., Sethuraman, J., Stein, C.: Better online buffer management. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 199–208 (2007)

Optimal Policy for Single-Machine Scheduling with Deterioration Effects, Learning Effects, Setup Times, and Availability Constraints

Sheng Yu[1,⋆], Yinfeng Xu[1,2], Ming Liu[3], and Feifeng Zheng[1,2]

[1] School of Management, Xi'an Jiaotong University, Xi'an, Shaanxi Province, 710049, P.R. China
[2] Ministry of Education Key Lab for Process Control and Efficiency Engineering, Xi'an, 710049, P.R. China
[3] School of Economics & Management, Tongji University, Shanghai, 200092, P.R. China

Abstract. In this paper, we introduce a single-machine scheduling model considering all of the following parameters: general deterioration and learning effects as well as general setup times. We prove that the shortest processing time (SPT) rule produces optimal schedules for the following seven minimization objectives: makespan, sum of αth ($\alpha \geq 0$) power of jobs' completion times, total weighted completion time, maximum lateness, total tardiness, total weighted tardiness, and number of tardy jobs. We further show that in the case of resumable scheduling with availability constraints, the above conclusions are still valid.

Keywords: Single-machine scheduling, Deterioration effects, Learning effects, Past-sequence-dependent setup times, Availability constraints.

1 Introduction

Scheduling a set of jobs on single or multiple machines is a classical problem, in which jobs are assumed to have fixed processing times [13]. Yet, there are numerous situations that the processing time deteriorates as the start time delays. A daily life example is to schedule maintenance or cleaning, in which a delay often requires additional effort to accomplish the task. Other examples can be found in fire fighting, steel production and financial management [11,12]. Scheduling of deteriorating jobs was first introduced by Browne and Yechiali [3], and Gupta and Gupta [9] independently. Another line is to consider learning effect of processing system. One system may gain more and more experience during processing jobs with similar characters, and then its speed of job processing may rise as the number of completed jobs increases. For the model with learning effect, the actual processing time of a job is a decreasing function with respect to its position in a processing sequence. Comprehensive surveys on these models have been given by Cheng, Ding and Lin [4], Biskup [2] and Gawiejnowicz [6].

⋆ Corresponding author, a.sheng@stu.xjtu.edu.cn

In the above papers on scheduling deteriorating jobs, the parameter of setup cost or setup times is often neglected, while it is reasonable and necessary in many scheduling businesses (see Allahverdi, Gupta and Aldowaisan [1]). There are two types of setup times: sequence-independent and sequence-dependent. The first type of setup time just depends on the task to be processed, regardless of its preceding tasks. While for the sequence-dependent type of setup time, it depends on both the current task and its preceding ones. Koulamas and Kyparisis [10] first introduced a scheduling case with past-sequence-dependent (p-s-d) setup times. They considered several objectives including makespan, total completion time, and total absolute differences in completion times and a bicriteria combination of the last two objectives. They proved that for single-machine scheduling with p-s-d setup times, the problems with the above objectives can be solved in polynomial time by a sorting procedure. Cheng, Lee and Wu [5] first addressed the problem of single-machine scheduling with p-s-d setup times, deterioration and learning effects simultaneously. Motivated by [5], we propose a model generalizing the forms of the functions mentioned as follows. Sun [14] introduced a new scheduling model, in which the actual processing time $p_{j[r]}$ is defined as an integrated function of learning and deterioration effects. More precisely, if j is scheduled in the rth position in a processing sequence, then $p_{j[r]} = p_j(1 + \sum_{l=1}^{r-1} p_{[l]})^{a_1} r^{a_2}$ where $a_1 \geq 1$ and $a_2 < 0$ denote deterioration and learning rates respectively. For the setup time of job j, i.e., $s_{j[r]}$, as proposed in Koulamas and Kyparisis [10] and Cheng, Lee and Wu [5], it depends on the information of those satisfied jobs before j, and it is said past-sequence-dependent (p-s-d). This is motivated from applications like high-tech manufacturing. In these two papers, $s_{j[r]}$ was defined as $s_{j[1]} = 0$ and $s_{j[r]} = \gamma \sum_{l=1}^{r-1} p^*_{[l]}$ where $\gamma > 0$ is a constant number.

By combining these models, we propose a general model with deterioration effects, learning effects and setup times. Here, the actual processing time of a job is a general function of the normal processing times of the jobs already processed and its scheduling position, as while as the p-s-d setup times of a job is a general function of the normal processing times of the jobs already processed. The proposed model is a generalized version of the Sun [14] and the Koulamas and Kyparisis [10] models, as discussed in the next section. The remainder of this paper is organized as follows. We introduce some notations and problem formulation in the next section. For this model, we prove the optimality of SPT for the following minimization objectives: makespan, sum of αth ($\alpha \geq 0$) power of jobs completion times, total weighted completion time, maximum lateness, total weighted tardiness, total tardiness, and number of tardy jobs. Moreover, the above conclusions are still valid even under resumable availability constraints.

The remainder of this paper is organized as follows. We introduce some notations and problem formulation in the next section. In Section 3 we prove that the shortest processing time (SPT) rule is optimal for several single-machine scheduling problems, and derive some corresponding corollaries in the case with availability constraints. The last section concludes this work.

2 Problem Definition and Notations

In this section, we introduce some notations to be used throughout the paper.

n	the number of jobs
p_j	the normal processing time of job j
d_j	the due date of job j
$p_{j[r]}$	the actual processing time of job j scheduled in the rth position
$p_{[r]}$	the normal processing time of a job scheduled in the rth position
$p^*_{[r]}$	the actual processing time of a job scheduled in the rth position
$s_{j[r]}$	the p-s-d setup time of job j scheduled in the rth position
C_j	the completion time of job j
L_j	the lateness of job j, i.e., $L_j = C_j - d_j$
T_j	the tardiness of job j, i.e., $T_j = \max\{L_j, 0\}$
$\sum U_j$	the number of tardy jobs
$C_{\max}$	the makespan, i.e., $C_{\max} = \max\{C_j \mid j = 1, 2, \cdots, n\}$
$\sum C_j$	the total completion time of all the jobs
$\sum C_j^\alpha$	the sum of the αth power of the job completion times of the jobs
$\sum w_j C_j$	the total weighted completion time of all the jobs
γ	the normalizing constant number with $\gamma \geq 0$

The formulation of the proposed problem is as follows. There is a single machine and n jobs to be processed on the machine. At most one job can be processed on the machine at a time. Each job j has a normal processing time p_j, a due date d_j and a weight w_j. Due to the deteriorating jobs with setup times and learning effect, the actual processing time and the p-s-d setup time of job j are $p_{j[r]} = f(p_j)(1 + \sum\limits_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}$ and $s_{j[r]} = \gamma \sum\limits_{l=1}^{r-1} g(p^*_{[l]})$ respectively, if it is scheduled in the rth position in a sequence where $a_1 \geq 1$ or $a_1 = 0$, $a_2 \leq 0$ and $s_{j[1]} = 0$. It is assumed that f and g are non-decreasing functions, and $f(x), g(x) > 0$ for $x > 0$.

If $f(p_j) = p_j$ and $\gamma = 0$, it is the Sun [14] model. On the other hand, if $f(p_j) = g(p_j) = p_j$ and $a_1 = a_2 = 0$, it is the Koulamas and Kyparisis [10] model. Throughout the paper, we use the three-field notation scheme $\alpha|\beta|\gamma$ introduced by Graham et al. [8]. We denote each scheduling model with objective F considered in this paper as $1|p_{j[r]} = f(p_j)(1 + \sum\limits_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}|F$ where $F \in \{C_{\max}, L_{\max}, \sum C_j^\alpha, \sum w_j C_j, \sum T_j, \sum w_j T_j, \sum U_j\}$.

3 Optimality of SPT

In this section ,we consider in total seven objectives. Before presenting the main results, we first state two lemmas which will be used later.

Lemma 1. *$(\theta + (1+\theta c_0)^{a_1} c_1^{a_2}) - (1 + \theta(1+c_0)^{a_1} c_1^{a_2}) \geq 0$, if $\theta \geq 1$, $a_1 \geq 1$ or $a_1 = 0$, $a_2 \leq 0$, $c_0 > 0$, and $c_1 > 1$.*

Proof. Let $h(\theta) = (\theta + (1+\theta c_0)^{a_1} c_1^{a_2}) - (1 + \theta(1+c_0)^{a_1} c_1^{a_2})$. When $a_1 = 0$, $h(\theta) = (\theta - 1)(1 - c_1^{a_2}) \geq 0$ for $\theta \geq 1$, $a_2 \leq 0$, and $c_1 > 1$.

When $a_1 \geq 1$, taking the first and second derivatives of $h(\theta)$ with respect to θ, for $\theta \geq 1$, $a_2 \leq 0$, $c_0 > 0$, and $c_1 > 1$ we have

$$h'(\theta) = 1 + a_1 c_0 (1+\theta c_0)^{a_1 - 1} c_1^{a_2} - (1+c_0)^{a_1} c_1^{a_2}, \quad \text{and}$$
$$h''(\theta) = a_1(a_1 - 1) c_0^2 (1+\theta c_0)^{a_1 - 2} c_1^{a_2} \geq 0.$$

Hence, $h'(\theta)$ is non-decreasing and $h'(\theta) \geq h'(1)$. Let $m(c_0) = h'(1)$. Taking the first derivative of $m(c_0)$ with respect to c_0, we have $m'(c_0) = a_1(a_1-1)c_0(1+c_0)^{a_1-2}c_1^{a_2} \geq 0$. Therefore, $h'(\theta) \geq m(c_0) > m(0) = 1 - c_1^{a_2} \geq 0$. Thus, $h(\theta) \geq h(1) = 0$. The lemma holds. □

Lemma 2. *$(\theta + x_1(1+\theta c_0)^{a_1} c_1^{a_2}) - (1 + \theta x_2 (1+c_0)^{a_1} c_1^{a_2}) \geq 0$, if $0 < x_2 \leq x_1 < 1$, $\theta \geq 1$, $a_1 \geq 1$ or $a_1 = 0$, $a_2 \leq 0$, $c_0 > 0$ and $c_1 > 1$.*

Proof. Let $h(\theta) = (\theta + x_1(1+\theta c_0)^{a_1} c_1^{a_2}) - (1 + \theta x_2(1+c_0)^{a_1} c_1^{a_2})$. When $a_1 = 0$, $h(\theta) = (\theta - 1) - (\theta x_2 - x_1) c_1^{a_2} \geq (\theta - 1) - (\theta x_1 - x_1) c_1^{a_2} = (\theta - 1)(1 - x_1 c_1^{a_2}) \geq 0$ for $0 < x_2 \leq x_1 < 1$, $\theta \geq 1$, $a_2 \leq 0$, and $c_1 > 1$.

When $a_1 \geq 1$, let $H(\theta) = (\theta + x_2(1+\theta c_0)^{a_1} c_1^{a_2}) - (1 + \theta x_2(1+c_0)^{a_1} c_1^{a_2})$, then $h(\theta) \geq H(\theta)$. Taking the first and second derivatives of $H(\theta)$ with respect to θ, we have

$$H'(\theta) = 1 + a_1 x_2 c_0 (1+\theta c_0)^{a_1-1} c_1^{a_2} - x_2(1+c_0)^{a_1} c_1^{a_2}, \quad \text{and}$$
$$H''(\theta) = a_1(a_1-1) x_2 c_0^2 (1+\theta c_0)^{a_1-2} c_1^{a_2} \geq 0.$$

Hence, similar to the proof of Lemma 1, we have $H(\theta) \geq 0$. Therefore, $h(\theta) \geq H(\theta) \geq 0$ for $a_1 \geq 1$.

The lemma holds. □

In the proofs of all the following theorems, we mainly adopt a basic method called adjacent job interchange technique. We first give some notations to be used in the proofs. Given an instance I, let S and S' be two job related schedules such that the difference between S and S' is a pairwise interchange of two adjacent jobs i and j. More precisely, let $S = (\pi\ i\ j\ \pi')$ and $S' = (\pi\ j\ i\ \pi')$, where $p_i \leq p_j$, and π and π' are partial sequences respectively. Either π or π' may be empty. We assume without loss of generality that there are $r - 1 (\geq 0)$ jobs in π. Thus, i and j are rth and $(r+1)$th jobs in S, respectively, whereas jobs j and i are scheduled in the rth and $(r+1)$th positions in S', respectively. Let $C(\pi)$ denote the completion time of the last job in π.

Theorem 1. *For problem $1|p_{j[r]} = f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}|C_{\max}$, an optimal schedule is obtained by sequencing jobs in the SPT order.*

Proof. Since $p_i \le p_j$, to prove that the SPT order is optimal, it suffices to prove that S dominates S', i.e., the $(r+1)$th jobs in S and S' satisfy $C_j(S) \le C_i(S')$. By definition, the completion times of jobs i and j in S and S' are given by, respectively,

$$\begin{aligned}
C_i(S) &= C(\pi) + f(p_i)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2} + \gamma \sum_{l=1}^{r-1} g(p^*_{[l]}),\\
C_j(S) &= C(\pi) + f(p_i)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2} + \gamma \sum_{l=1}^{r-1} g(p^*_{[l]})\\
&\quad + f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}) + f(p_i))^{a_1}(r+1)^{a_2}\\
&\quad + \gamma[\sum_{l=1}^{r-1} g(p^*_{[l]}) + g(f(p_i)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2})], \qquad (1)\\
C_j(S') &= C(\pi) + f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2} + \gamma \sum_{l=1}^{r-1} g(p^*_{[l]}), \text{ and}\\
C_i(S') &= C(\pi) + f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2} + \gamma \sum_{l=1}^{r-1} g(p^*_{[l]})\\
&\quad + f(p_i)(1+\sum_{l=1}^{r-1} f(p_{[l]}) + f(p_j))^{a_1}(r+1)^{a_2}\\
&\quad + \gamma[\sum_{l=1}^{r-1} g(p^*_{[l]}) + g(f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2})]. \qquad (2)
\end{aligned}$$

Taking the difference between (1) and (2), it is obtained that

$$\begin{aligned}
&C_i(S') - C_j(S)\\
&= (f(p_j) - f(p_i))(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2} + f(p_i)(1+\sum_{l=1}^{r-1} f(p_{[l]}) + f(p_j))^{a_1}(r+1)^{a_2}\\
&\quad - f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}) + f(p_i))^{a_1}(r+1)^{a_2}\\
&\quad +\gamma[g(f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}) - g(f(p_i)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2})]. \qquad (3)
\end{aligned}$$

By substituting $\theta = \frac{f(p_j)}{f(p_i)}$, $c_0 = \frac{f(p_i)}{1+\sum_{l=1}^{r-1} f(p_{[l]})}$, and $c_1 = \frac{r+1}{r}$ into (3), and simplifying, we obtain

$$
\begin{aligned}
&C_i(S') - C_j(S) \\
&= f(p_i)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2} \{(\theta + (1 + \theta c_0)^{a_1} c_1^{a_2}) - (1 + \theta(1 + c_0)^{a_1} c_1^{a_2})\} \\
&+\gamma[g(f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}) - g(f(p_i)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2})]. \qquad (4)
\end{aligned}
$$

Since the first term on the right-hand-side of (4) is non-negative by lemma 1. For $p_j - p_i \geq 0$, we have $f(p_j) - f(p_i) \geq 0$ and $f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2} \geq f(p_i)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}$. Since $\gamma \geq 0$ and $g(x)$ is a non-decreasing function, the second term is non-negative. Therefore, we have $C_i(S') - C_j(S) \geq 0$.

Thus, S dominates S'. Therefore, repeating this interchange argument for all the jobs not sequenced according to the SPT rule will yield the theorem. □

Corollary 1. *For problem* $1|d_j = d, p_{j[r]} = f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}| \sum U_j$, *an optimal schedule is obtained by sequencing jobs in the SPT order.*

Proof. According to the SPT rule, for an arbitrary job instance, the number of completed jobs in the schedule by the SPT rule is not less than that of any other schedule at any time. Thus, the corollary is straightforward. □

Theorem 2. *For problem* $1|p_{j[r]} = f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}| \sum C_j^{\alpha}$ *where* $\alpha \geq 0$ *is a constant number, an optimal schedule is obtained by sequencing jobs in the SPT order.*

Proof. The completion time of job i in S and the completion time of job j in S' are shown as Theorem 1. Since $p_j \geq p_i$ and $f(x)$ is a non-decreasing function, we have

$$C_j(S') - C_i(S) = (f(p_j) - f(p_i))(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2} \geq 0 \qquad (5)$$

From the proof of Theorem 1, we know $C_j(S) \leq C_i(S')$. Since the completion time of a job is always at least zero, we can get (i) $C_j^{\alpha}(S) \leq C_i^{\alpha}(S')$, (ii) $C_i^{\alpha}(S) \leq C_j^{\alpha}(S')$. Thus, $C_i^{\alpha}(S) + C_j^{\alpha}(S) \leq C_j^{\alpha}(S') + C_i^{\alpha}(S')$ which shows that S dominates S'. This completes the proof of the theorem. □

For any two jobs i and j in an instance I, if $p_i \leq p_j$ implies $w_i \geq w_j$, we say the jobs' processing times and the weights are *agreeable*.

Theorem 3. *For problem* $1|p_{j[r]} = f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}| \sum w_j C_j$, *if the jobs have agreeable weights, an optimal schedule can be obtained by sequencing the jobs in the SPT order.*

Proof. Since $p_i \leq p_j$, which implies $w_i \geq w_j$. In order to show S dominates S', it suffices to show that (i) $C_j(S) \leq C_i(S')$, (ii) $w_i C_i(S) + w_j C_j(S) \leq w_j C_j(S') + w_i C_i(S')$. The proof of part (i) is given in Theorem 1. Therefore, we present the proof of part (ii) as follows.

Under S and S', the completion times of jobs i and j are shown as Theorem 1. Thus, we have

$$\begin{aligned}
&\{w_j C_j(S') + w_i C_i(S')\} - \{w_i C_i(S) + w_j C_j(S)\} \\
&= (f(p_j) - f(p_i))(w_i + w_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2} \\
&+ w_i f(p_i)(1 + \sum_{l=1}^{r-1} f(p_{[l]}) + f(p_j))^{a_1}(r+1)^{a_2} - w_j f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}) + f(p_i))^{a_1}(r+1)^{a_2} \\
&+ w_i w_j \gamma\{\frac{w_i - w_j}{w_i w_j} \sum_{l=1}^{r-1} g(p^*_{[l]}) + (\frac{1}{w_j} - \frac{1}{w_i})[g(f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}) \\
&- g(f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2})]\}.
\end{aligned} \tag{6}$$

Let $x_1 = \frac{w_i}{w_i + w_j}$, $x_2 = \frac{w_j}{w_i + w_j}$, $\theta = \frac{f(p_j)}{f(p_i)}$, $c_0 = \frac{f(p_i)}{1 + \sum_{l=1}^{r-1} f(p_{[l]})}$, and $c_1 = \frac{r+1}{r}$. Hence, $0 < x_2 \leq x_1 < 1$, $\theta \geq 1$, $c_0 > 0$ for $f(p_j) \geq f(p_i) > 0$, $w_i \geq w_j > 0$, and $c_1 > 1$. Thus, the first three terms on the right-hand-side of (6) can be rewritten as

$$(w_i + w_j) f(p_i)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}((\theta + x_1(1 + \theta c_0)^{a_1} c_1^{a_2}) - (1 + \theta x_2(1 + c_0)^{a_1} c_1^{a_2})).$$

The above expression is non-negative by Lemma 2. Since $w_i \geq w_j$ and $\frac{1}{w_i} \leq \frac{1}{w_j}$, similar to the proof of Theorem 2 the last term on the right-hand-side of (6) is also non-negative. Therefore, we have $w_i C_i(S) + w_j C_j(S) \leq w_j C_j(S') + w_i C_i(S')$.

This completes the proof of part (ii) and thus of the theorem. □

For any two jobs i and j in an instance I, if $p_i \leq p_j$ implies $d_i \leq d_j$, we say the jobs' processing times and the due dates are *disagreeable*.

Theorem 4. *For problem* $1|p_{j[r]} = f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}|L_{\max}$, *an optimal schedule is obtained by sequencing jobs in the SPT order if the job processing times and the due dates are disagreeable.*

Proof. Hence $d_i \leq d_j$ for $p_i \leq p_j$. From the proof of Theorem 1, we have $C_j(S') \leq C_i(S')$. Since $d_i \leq d_j$, we obtain that $L_j(S') \leq L_i(S')$. In order to show S dominates S', it suffices to show that $\max\{L_i(S), L_j(S)\} \leq L_i(S')$. Since $C_j(S) \leq C_i(S')$ and $C_i(S) \leq C_i(S')$ by the proof of Theorem 1, together with $d_i \leq d_j$, we have that $L_j(S) \leq L_i(S')$ and $L_i(S) \leq L_i(S')$.

Therefore, $\max\{L_i(S), L_j(S)\} \leq \max\{L_j(S'), L_i(S')\}$. This completes the proof. □

Theorem 5. *For problem* $1|p_{j[r]} = f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}| \sum w_j T_j$, *an optimal schedule is obtained by sequencing jobs in the SPT order if the job processing times and the due dates are disagreeable while the jobs have agreeable weights at the same time.*

Proof. By the assumption in the theorem, if $p_i \leq p_j$, then $w_i \geq w_j$ and $d_i \leq d_j$. In order to show that S dominates S', it suffices to show that $w_i T_i(S) + w_j T_j(S) \leq w_j T_j(S') + w_i T_i(S')$. We consider two cases.

Case 1. $C_j(S') \leq d_j$. By definition, the weighted tardiness of jobs i and j in S and in S' are

$$w_i T_i(S) + w_j T_j(S) = w_i \max\{C_i(S) - d_i, 0\} + w_j \max\{C_j(S) - d_j, 0\}, \text{ and}$$
$$w_j T_j(S') + w_i T_i(S') = w_i \max\{C_i(S') - d_i, 0\}.$$

Suppose that neither $T_i(S)$ nor $T_j(S)$ is zero since otherwise we already have the desired result. Then,

$$\begin{aligned}&\{w_j T_j(S') + w_i T_i(S')\} - \{w_i T_i(S) + w_j T_j(S)\}\\ &= (w_j d_j + w_i C_i(S')) - (w_i C_i(S) + w_j C_j(S))\\ &\geq \{w_j C_j(S') + w_i C_i(S')\} - \{w_i C_i(S) + w_j C_j(S)\}.\end{aligned}$$

By Theorem 3, $\{w_j T_j(S') + w_i T_i(S')\} \geq \{w_i T_i(S) + w_j T_j(S)\}$ in this case.

Case 2. $C_j(S') > d_j$. By definition, the weighted tardiness of jobs i and j in S and in S' are

$$w_i T_i(S) + w_j T_j(S) = w_i \max\{C_i(S) - d_i, 0\} + w_j \max\{C_j(S) - d_j, 0\}, \text{ and}$$
$$w_j T_j(S') + w_i T_i(S') = w_j (C_j(S') - d_j) + w_i \max\{C_i(S') - d_i, 0\}.$$

Suppose neither $T_i(S)$ nor $T_j(S')$ is zero. If not, it is easy to get the result.

$$\begin{aligned}&\{w_j T_j(S') + w_i T_i(S')\} - \{w_i T_i(S) + w_j T_j(S)\}\\ &= \{w_j C_j(S') + w_i C_i(S')\} - \{w_i C_i(S) + w_j C_j(S)\} \geq 0.\end{aligned}$$

This completes the proof of theorem. □

Corollary 2. *For problem* $1|p_{j[r]} = f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}| \sum T_j$, *an optimal schedule is obtained by sequencing jobs in the SPT order if the job processing times and the due dates are disagreeable.*

In the following we consider another situation where the machine is not continuously available, i.e., there are $h \geq 1$ disjoint unavailable periods. Denote these periods as time intervals $[e_{k,1}, e_{k,2}]$ where $e_{k,1} < e_{k,2}$ for $1 \leq k \leq h$, and $e_{0,1} = e_{0,2} = 0$ (see Gawiejnowicza and Kononovb [7]). We focus on resumable model such that a job interrupted by an unavailable period will be processed from where it was preempted at the end of the unavailable period. Note that one job may be interrupted by more than one unavailable periods.

Given one instance I, let σ and S denote the schedules obtained by the SPT rule for problems $1|h, res, p_{j[r]} = f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}|F$ and $1|p_{j[r]} = f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}|F$, respectively. As a consequence, we know that the job processing sequence of σ is the same as that of S. Let $C_{j[l]k}$ denote the completion time of job j which is scheduled in the lth position in σ and satisfies $e_{k,1} < C_{j[l]k}(\sigma) \leq e_{k,2}$ for some k. So,

$$C_{j[l]k}(\sigma) = C_{j[l]}(S) + \sum_{i=0}^{k} (e_{i,2} - e_{i,1}),$$

and

$$\sum_{i=0}^{k-1} (e_{i+1,1} - e_{i,2}) < C_{j[l]}(S) \leq \sum_{i=0}^{k} (e_{i+1,1} - e_{i,2}).$$

Corollary 3. *For problem $1|h, res, p_{j[r]} = f(p_j)(1+\sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}|C_{\max}$, an optimal schedule is obtained by sequencing jobs in the SPT order.*

Proof. We still use the same notations about sets S and S' in Theorem 1. Similar to the definition of S', σ' is defined to be an (i, j)-pairwise interchanged variant of σ. From Theorem 1, we have $C_i(S') \geq C_j(S)$, which implies $C_{ik'}(\sigma') \geq C_{jk}(\sigma)$ where $k' \geq k$. This completes the proof. $\square$

Similar to the proofs of Theorems 2 to 5, for the case with availability constraints, the SPT rule is still optimal for problem $1|h, res, p_{j[r]} = f(p_j)(1 + \sum_{l=1}^{r-1} f(p_{[l]}))^{a_1} r^{a_2}, s_{psd}|F$, where p_j , w_j and d_j have the same agreeable and disagreeable properties as for the previous theorems and corollaries.

4 Conclusion

In this paper, we studied single-machine scheduling with general deterioration and learning effects as well as general p-s-d setup times. We proved that for seven minimization objectives such as makespan, sum of αth ($\alpha \geq 0$) power of jobs' completion times, etc., the SPT rule produces optimal schedules. We further extend the results to the case with unavailable periods and resumable preemption. Future research may focus on analyzing the above minimization objectives in parallel machines or flow shop, and other environments.

Acknowledgments. This work is partially supported by NSF of China under Grants 71071123, 60736027 and 60921003.

References

1. Allahverdi, A., Gupta, J.N.D., Aldowaisan, T.: A review of scheduling research involving setup considerations. Omega 27, 219–239 (1999)
2. Biskup, D.: A state-of-the-art review on scheduling with learning effects. European Journal of Operational Research 188, 315–329 (2008)
3. Browne, S., Yechiali, U.: Scheduling deteriorating jobs on a single processor. Operations Research 38(3), 495–498 (1990)
4. Cheng, T.C.E., Ding, Q., Lin, B.M.T.: A concise survey of scheduling with time-dependent processing times. European Journal of Operational Research 152, 1–13 (2004)
5. Cheng, T.C.E., Lee, W.C., Wu, C.C.: Scheduling problems with deteriorating jobs and learning effects including proportional setup times. Computers and Industial Engineering 58, 326–331 (2010)
6. Gawiejnowicz, S.: Time-Dependent Scheduling. Springer, Berlin (2008)
7. Gawiejnowicza, S., Kononovb, A.: Complexity and approximability of scheduling resumable proportionally deteriorating jobs. European Journal of Operational Research 200, 305–308 (2010)
8. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 5, 287–326 (1979)
9. Gupta, J.N.D., Gupta, S.K.: Single facility scheduling with nonlinear processing times. Computers and Industrial Engineering 14(4), 387–393 (1988)
10. Koulamas, C., Kyparisis, G.J.: Single-machine scheduling problems with past-sequence-dependent setup times. European Journal of Operational Research 187, 1045–1049 (2008)
11. Kunnathur, A.S., Gupta, S.K.: Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. European Journal of Operation Research 47(1), 56–64 (1990)
12. Mosheiov, G.: Scheduling jobs under simple linear deterioration. Computers and Operations Research 21(6), 653–659 (1994)
13. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Prentice-Hall, Upper Saddle River (2002)
14. Sun, L.: Single-machine scheduling problems with deteriorating jobs and learning effects. Computers and Industial Engineering 57, 843–846 (2009)

Algebraic Algorithm for Scheduling Data Retrieval in Multi-channel Wireless Data Broadcast Environments*

Xiaofeng Gao[1], Zaixin Lu[2], Weili Wu[2], and Bin Fu[3]

[1] Department of Computer Science and Engineering,
Shanghai JiaoTong University, Shanghai, P.R. China
`xgao@cs.sjtu.edu.cn`

[2] Department of Computer Science,
University of Texas at Dallas, Richardson, TX 75080, USA
{`zaixinlu,weiliwu`}`@utdallas.edu`

[3] Department of Computer Science,
University of Texas–Pan American, Edinburg, TX 78539, USA
`binfu@cs.panam.edu`

Abstract. Due to more and more customers keen on mobile services, we may face the mobile network congestion problem. Therefore, it is necessary to develop new data retrieval method to provide users with reliable and timely access to the data scourers. In this paper, we study the scheduling problem for retrieving data from multi-channel data broadcast environments. In general conditions, the most important two issues for queries in mobile computing systems are the energy cost and the query response time. In order to improve the query efficiency, we develop a randomized algebraic algorithm that takes both energy cost and access time into consideration to schedule the data retrieval process in multi-channel environments. It can be used in almost any broadcast environment, in which the data access frequencies, data sizes, and channel bandwidths can all be non-uniform.

Keywords: Wireless data broadcast, Multi-channel, Mobile computing, Data retrieval optimization.

1 Introduction

Wireless data broadcast is very suitable for disseminating public information to large number of mobile users. Generally, there are two major measures when evaluating the query efficiency in such environments. One is *access time* and the other is *energy efficiency*. The *access time* denotes the time interval between the moment a query starts to the moment all the requested data have been downloaded. Obviously, users prefer short *access time*. In addition, the power

* This work is supported in part by National Science Foundation under grant CNS1016320 and CCF0829993.

supply is finite for mobile devices, which means *energy efficiency* is also very important when design data retrieval method in mobile computing environments.

In a general data broadcast network, a base station disseminates data to mobile users via one or multiple channels; so there are totally two entities: *mobile client* and *base station*. Fig. 1 shows a typical broadcast network. A base station can send information via radio waves to large number of mobile clients simultaneously, and the cost at the server site will not change as the number of clients increases. For instance, the base station B_2 in Fig. 1 provide services to more users, but its costs is nearly the same as that of B_1. Moreover, the *mobile client* may has two modes: *active mode* and *sleep mode*. With the help of index, clients can get the arriving time of their requested data in advance and "sleep" to save energy when there are no data of interests.

Although, it is relatively easy for data retrieving if all the data are scheduled on one channel, users may prefer partition the data onto multiple channels to reduce the average expected access time. But it should be noticed that if a client is downloading a data from channel c_i at time t_0, then it cannot switch to channel c_j, where $j \neq i$, to download another data at time $t_0 + 1$. The reason is that switching the channels takes time, and if a client want to download data from another channel, it needs one time slot for channel switching. Fig. 2 gives a typical process of data retrieval in multi-channel broadcast environments. The query data set is $\{d_1, d_3, d_5\}$, and a user can download data object d_1 and d_3 from channel c_1, and then switch to channel c_3 at time 6 to download data object d_5 at time 7. However, after time 5, the user cannot switch from channel c_1 to c_2 to download data d_5 at time 6. From Fig. 2, we also can get that the bandwidths of different channels are non-necessarily the same. Actually, the bandwidth of channel c_2 is twice as that of c_1 or c_3, thus d_3 or d_5, which take two time slots on c_1 or c_3, can be broadcasted in one time slot by c_2.

In most situations, it is much more likely a client query a set of data instead of only one data at a time. After obtaining the locations of requested data items, we need to make a schedule to download the data one by one in some order.

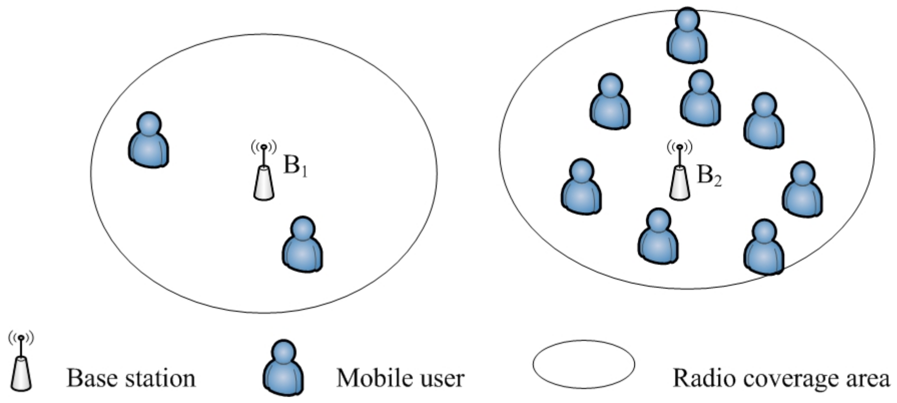


Fig. 1. Data Broadcast Network

An unwise retrieving schedule may result in long access time and unnecessary energy consumption. Usually, the energy consumption is evaluated base on the following two metrics: 1) tune-in time and 2) the number of channel switchings. Assume the arriving time of each requested data item is already known from the index, then the energy consumption depends purely on the number of channel switching happens during the retrieval process. In this paper we propose a randomized algebraic algorithm to reduce the access time and channel switchings for data retrieval in multi-channel environments. It can be used in almost any data broadcast programs, in which the data access frequencies, data sizes, and channel bandwidths can all be non-uniform.

The remainder of this paper is organized as follows. Section 2 presents the related works to wireless data broadcast. In section 3, we give an algebraic algorithm that considers both access latency and energy cost to get optimal solutions for data retrieving in multi-channel environments. Finally, in section 4, we conclude this paper.

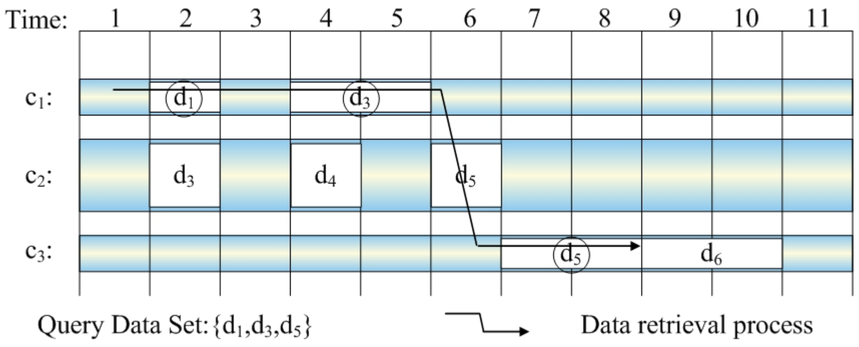


Fig. 2. Data Retrieval Process

2 Previous Works

Scheduling is an important issue in the area of wireless data broadcast. Acharya et al. first proposed the scheduling problem for data broadcast [1], and Prabhakara et al. suggested the multi-channel model for data broadcast to improve the data delivery performance [2]. After that, many works have been done for scheduling data on multiple channels to reduce the expected access time [5,6,12]. Besides, some researches began to study how to allocate dependent data on broadcast channels. (see e.g. [14,10,11]). With respect to index, many methods have been proposed to improve the search efficiency in data broadcast systems (see e.g. [8,3,9,10,11]). Furthermore, Jung et al. proposed a tree-structured index algorithm that allocates indices and data on different channels [7]. Lo and Chen designed a parameterized schema for allocating indices and data optimally on multiple channels such that the average expected access latency is minimized [4].

In terms of data retrieval scheduling, Hurson et al. proposed two heuristic algorithms for downloading multiple data items from multiple channels [15]. Shi et al. investigated how to schedule multiple processes to download a set of data items [16]. Both of them investigate the data retrieval problem by assuming that the data are allocated on multiple channels without replication. However, as shown in the prior studies [1,20,21,22], employing data replication in data broadcast programs will reduce the expected access time. Fig. 3 shows why disseminating replicative data by multiple channels can reduce both access time and energy consumption. The first program allocates data without replication. d_1 and d_2 are separately scheduled on channels c_1 and c_2. we can download d_1 or d_2 in one time slot, but we need at least 3 time slots and 1 channel switching to download both d_1 and d_2 in such a system. If allocating data on channels like the way of program 2, we can still retrieve each datum in one time slot and we can retrieve both of them in 2 time slots without channel switching.

In this paper, we develop an algebraic algorithm for solving the problem of retrieving multiple data from multiple channels, in which the data can be non-uniform length and are replication-allowed to be broadcasted via multiple channels.

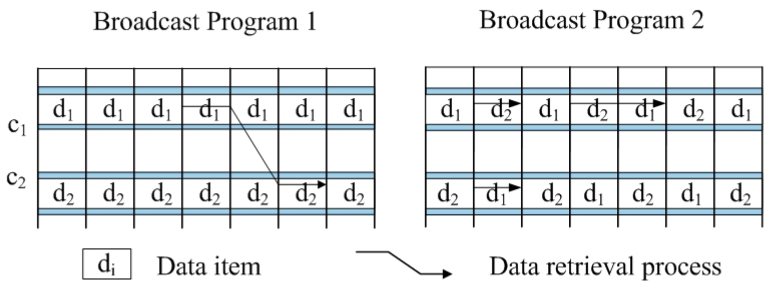


Fig. 3. Two Types of Broadcasting Database

3 Methodology

In this section we present an algebraic algorithm for the data retrieval problem in multi-channel environments. It can detect if a given problem has a schedule to download all the requested data before time t and with at most h channel switchings in $O(2^k(hnt)^{O(1)})$ time, where n is the number of channels and k is the number of required data items. We first give a problem definition in section 3.1, and then present the algorithm in section 3.2.

3.1 Problem Description

Let's consider a mobile user wants to query a set of data $D = \{d_1, d_2, \cdots, d_k\}$, which are broadcasted via channels in $C = \{c_1, c_2, \cdots, c_n\}$. We assume the locations of all the data in D are known; and we also assume each channel is partitioned into discrete time slots and one time slot is the smallest unit for storing data. Let a tuple $s = \{i_s, j_s, t_s, t'_s\}$ denote the data d_{i_s} can be downloaded from channel c_{j_s} during the time span $[t_s, t'_s]$, then it is clear that a valid data retrieval schedule is a sequence of k intervals $s_1, s_2, \cdots, s_k$, each tuple corresponds to a distinct data item in D and there is no conflicts between any two of the k tuples.

A Decision Problem: Given a data set D, a channel set C, a time threshold t and a switching threshold h, find a valid data retrieval schedule to download all the data in D from C before time t with at most h switchings.

To solve the above decision problem, we developed a randomized algebraic algorithm. We present it in detail next.

3.2 Algorithm

The basic idea of our algebraic algorithm is that for each data item $d_i \in D$, where D is the query data set, we create a variable x_i to represent it. Therefore, given $D = \{d_1, d_2, \cdots, d_k\}$, we construct a variable set $X = \{x_1, x_2, \cdots, x_k\}$. We then design a circuit $H_{t,h,n}$ such that a schedule without conflict will be generated by a multilinear monomial in the sum of product expansion of the circuit. A multilinear monomial is a monomial such that each variable has degree exactly one, for examples, $x_3x_5x_6$ is a multilinear monomial, but $x_3x_5^3x_6^2$ is not. The existence of schedules to download all the data items in D from the multiple channels of C is converted into the existence of multilinear monomials of $H_{t,h,n}$. Replace each variable by a specified binary vector can remove all of the non-multilinear monomials by converting them to zero. Thus, the data retrieval problem is transformed into testing if a multivariate polynomial is zero. It is well known that randomized algorithms can be used to check if a circuit is identical to zero in polynomial time.

Lemma 1. *There is a polynomial time algorithm such that given a channel c_i, a time interval $[t_1, t_2]$, and an integer m, it constructs a circuit of polynomial $P_{i,t_1,t_2,m}$ such that for any subset $D' = \{d_{i_1}, \cdots, d_{i_m}\} \subseteq D$ which has a size of m and is downloadable in the time interval $[t_1, t_2]$ from channel c_i, the product expansion of $P_{i,t_1,t_2,m}$ contains a multilinear monomial $x_{i_1}x_{i_2}\cdots x_{i_m}$.*

Proof. We can use a recursive way to compute the circuit $P_{i,t_1,t_2,m}$ in polynomial time.

1. $P_{i,t_1,t_2,0} = 0$.
2. $P_{i,t_1,t_2,1} = \sum_j x_j$, $x_j \subseteq X$ and the corresponding datad_j is entirely in the time interval $[t_1, t_2]$ of channel c_i.
3. $P_{i,t_1,t_2,l+1} = \sum_j x_j \cdot P_{i,t_1,t_2',l} + P_{i,t_1,t_2',l+1}$, d_j starts at time $t_2' + 1$ and ends before time t_2 on channel c_i.

When computing $P_{i,t_1,t_2,l+1}$, x_j multiplies $P_{i,t_1,t_2',l}$ is based on the case that d_j is downloadable from time $t_2' + 1$ to t_2 in the final phase, and the other l data items are downloadable before time t_2'. The term $P_{i,t_1,t_2',l+1}$ is the case that $l+1$ items are downloaded before time t_2'. Note that the parameter m in $P_{i,t1,t2,m}$ controls the total number of data to be downloaded.

Definition 1. *A subset data items $D' = \{d_{i_1}, \cdots, d_{i_m}\} \subseteq D$ is (i,t,h)-downloadable if we can download all data items in D' before time t, the total number of channel switches is at most h, and the last downloaded item is from channel c_i.*

Lemma 2. *Given two integers t and h, there is a polynomial time algorithm to construct a circuit of polynomial $F_{i,t,h,m}$ such that for any (i,t,h)-downloadable subset $D' = \{d_{i_1}, \cdots, d_{i_m}\} \subseteq D$, the product expansion of $F_{i,t,h,m}$ contains a multilinear monomial $(x_{i_1}, \cdots, x_{i_m})Y$, where Y is a multilinear monomial doesn't include any variable in X.*

Proof. We still use a recursive way to construct the circuit. Some additional variables are used as needed. Without loss of generality, we assume the data retrieval process start at time 0.

1. $F_{i,t,0,0} = 0$.
2. $F_{i,t,0,1} = P_{i,1,t,1} \cdot y_{i,t,0,1}$.
3. $F_{i,t,h'+1,m'+1} = y_{i,t,h'+1,m'+1,0}(\sum_{t'<t} F_{i,t',h'+1,m'} \cdot P_{i,t'+1,t,1})$
   $+ y_{i,t,h'+1,m'+1,1}(\sum_{j \neq i} \sum_{t'<t} F_{i,t'-1,h',m'} \cdot P_{i,t'+1,t,1})$

The computing of $F_{i,t,h'+1,m'+1}$ is based on two cases, and we use two variables, $y_{i,t,h'+1,m'+1,0}$ and $y_{i,t,h'+1,m'+1,1}$, to mark them respectively. We now present an algorithm that involves one layer randomization to determine if there is a schedule to download all the data items in D before time t and with at most h channel switchings.

Theorem 1. *There is an $O(2^k(hnt)^{O(1)})$ time randomized algorithm to determine if there is a scheduling to download $k = |D|$ data items before time t and the number of channel switches is at most h, where n is the total number of channels.*

Proof. By Lemma 2, we can construct a circuit $H_{t,h,n} = \sum_{i=1}^{n} F_{i,t,h,k}$ in polynomial time. It is easy to see there is a scheduling for downloading the k data items before time t and with h channel switches, if and only if the sum product expansion of $H_{t,h,n}$ has a multilinear monomial $(x_1, \cdots, x_k)Y$.

Replace each x_i by a vector $w_i = w_0^T + v_i^T$, where w_0 is the all-zeros vector of dimension k, and v_i is a binary vector of dimension k with its i^{th} element is 1 and all other elements are 0. Assume $k = 3$, we define the following operations:

$$v_a \cdot v_b = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} (a_1+b_1)(mod 2) \\ (a_1+b_2)(mod 2) \\ (a_1+b_3)(mod 2) \end{pmatrix} \tag{1}$$

$$(v_a + v_b) \cdot v_c = v_a \cdot v_c + v_b \cdot v_c \tag{2}$$

By Equation 1 and 2, for any k-dimensional binary vector $w' = w_0 + v$, we have $w'^2 = w_0^2 + 2w_0 \cdot v + v^2 = w_0 + 2(w_0 \cdot v) + w_0 = 2(w_0 \cdot v) + 2w_0 = 0$, because of the coefficients are in the field of G_2. The replacement $x_i = w_i(i = 1, \cdots, m)$ make all the non-multilinear monomials become zero. Meanwhile, all the multilinear monomials remain non-zero. Hence, it is clear that there is a scheduling to download all the data items in D before time t and with at most h channel switchings if and only if $H_{t,h,n|x_i=w_i(i=1,\cdots,k)}$ is a non-zero polynomial. The variables in Y makes it impossible to have cancelation when adding two identical multilinear monomials, which can be generated from different paths with variables in $\{x_1, \cdots, x_k\}$. It is well known that randomized algorithms can be used to check if a circuit is identical to zero in polynomial time [17], [18].

The algorithm generates less than 2^k terms during the computing process since there are at most 2^k distinct binary vectors. Therefore, the computational time is $O(2^k(nht)^{O(1)})$.

4 Conclusions

In this paper, we take both access time and channel switchings into consideration to investigate the minimum cost data retrieval problem in multi-channel data broadcast environments. The algorithm proposed can detect if a given data retrieval problem has a solution with access time t and number of switchings h in $O(2^k(hnt)^{O(1)})$ time, where n is the number of channels and k is the number of requested data items. It can be used in almost any broadcast environment, in which the data access frequencies, data sizes, and channel bandwidths can all be non-uniform.

References

1. Acharya, S., Alonso, R., Franklin, M., Zdonik, S.: Broadcast Disks: Data Management for Asymmetric Communication Environments. In: The 1995 ACM Special Interest Group on Management of Data Conference, pp. 199–210 (1995)
2. Prabhakara, K., Hua, K.A., Oh, J.: Multi-Level Multi-Channel Air Cache Designs for Broadcasting in a Mobile Environment. In: The 2000 International Conference on Data Engineering, pp. 167–176 (2000)
3. Shivakumar, N., Venkatasubramanian, S.: Efficient Indexing for Broadcast based Wireless Systems. ACM/Baltzer Mobile Network and Application 1(4), 433–446 (1996)
4. Lo, S.C., Chen, A.L.P.: Optimal Index and Data Allocation in Multiple Broadcast Channels. In: The 2000 International Conference on Data Engineering, pp. 293–302 (2000)
5. Yee, W.G., Navathe, S.B., Omiecinski, E., Jermaine, C.: Efficient Data Allocation over Multiple Channels at Broadcast Servers. IEEE Transactions on Computers 51(10), 1231–1236 (2002)
6. Zheng, B., Wu, X., Jin, X., Lee, D.L.: Tosa: a Near-Optimal Scheduling Algorithm for Multi-Channel Data Broadcast. In: The 2005 International Conference on Mobile Data Management, pp. 29–37 (2005)
7. Jung, S., Lee, B., Pramanik, S.: A Tree-Structured Index Allocation Method with Replication over Multiple Broadcast Channels in Wireless Environment. IEEE Transaction on Knowledge and Data Engineering 17(3), 311–325 (2005)

8. Imielinski, T., Viswanathan, S., Badrinath, B.R.: Data on Air: Organization and Access. IEEE Transactions on Knowledge and Data Engineering 9, 353–372 (1996)
9. Xu, J., Lee, W.C., Tang, X., Gao, Q., Li, S.: An Error-Resilient and Tunable Distributed Indexing Scheme for Wireless Data Broadcast. IEEE Transactions on Knowledge and Data Engineering 18(3), 392–404 (2006)
10. Yao, Y., Tang, X., Lim, E.P., Sun, A.: An Energy-Efficient and Access Latency Optimized Indexing Scheme for Wireless Data Broadcast. IEEE Transactions on Knowledge and Data Engineering 18(8), 1111–1124 (2006)
11. Zheng, B., Lee, W.C., Liu, P., Lee, D.L., Ding, X.: Tuning On-Air Signatures for Balancing Performance and Confidentiality. IEEE Transactions on Knowledge and Data Engineering 21(12), 1783–1797 (2009)
12. Ardizzoni, E., Bertossi, A.A., Ramaprasad, S., Rizzi, R., Shashanka, M.V.S.: Optimal Skewed Data Allocation on Multiple Channels with Flat Broadcast per Channel. IEEE Transactions on Computers 54(5), 558–572 (2005)
13. Huang, J.L., Chen, M.S.: Dependent Data Broadcasting for Unordered Queries in a Multiple Channel Mobile Environment. IEEE Transactions on Knowledge and Data Engineering 16(9), 1143–1156 (2004)
14. Lee, G., Yeh, M.S., Lo, S.C., Chen, A.: A Strategy for Efficient Access of Multiple Data Items in Mobile Environments. In: The 2002 International Conference on Mobile Data Management, pp. 71–78 (2002)
15. Hurson, A.R., Munoz-Avila, A.M., Orchowski, N., Shirazi, B., Jiao, Y.: Power Aware Data Retrieval Protocols for Indexed Broadcast Parallel Channels. Pervasive and Mobile Computing 2(1), 85–107 (2006)
16. Shi, Y., Gao, X., Zhong, J., Wu, W.: Efficient Parallel Data Retrieval Protocols with MIMO Antennae for Data Broadcast in 4G Wireless Communications. In: The 2010 International Conference on Database and Expert Systems Applications (2010)
17. Williams, R.: Finding Paths of Length k in $O*(2^k)Time$. Information Processing Letters 109(6), 315–318 (2009)
18. Koutis, I.: Faster Algebraic Algorithms for Path and Packing Problems. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 575–586. Springer, Heidelberg (2008)
19. Huang, J.L., Chen, M.S., Peng, W.C.: Broadcasting Dependent Data for Ordered Queries without Replication in a Multi-Channel Mobile Environment. In: The 2003 International Conference on Data Engineering, pp. 692–694 (2003)
20. Huang, J.L., Chen, M.S.: Broadcast Program Generation for Unordered Queries with Data Replication. In: The 2003 ACM Symposium on Applied Computing, pp. 866–870 (2003)
21. Foltz, K., Xu, L., Bruck, J.: Scheduling for Efficient Data Broadcast over Two Channels. In: The 2004 International Symposium on Information Theory, pp. 113–116 (2004)
22. Lu, Z., Shi, Y., Wu, W., Fu, B.: Efficient Data Retrieval Scheduling for Multi-Channel Wireless Data Broadcast. IEEE Transaction on Knowledge and Data Engineering (submitted to 2011)

Hamiltonian Cycles through Prescribed Edges in k-Ary n-Cubes

Iain A. Stewart

School of Engineering and Computing Sciences, Durham University,
Science Labs, South Road, Durham DH1 3LE, U.K.

Abstract. We prove that if P is a set of at most $2n-1$ edges in a k-ary n-cube, where $k \geq 4$ and $n \geq 2$, then there is a Hamiltonian cycle on which every edge of P lies if, and only if, the subgraph of the k-ary n-cube induced by the edges of P is a vertex-disjoint collection of paths. This answers a question posed by Wang, Li and Wang who proved the analogous result for 3-ary n-cubes.

Keywords: Hamiltonian cycles, k-ary n-cubes, prescribed edges.

1 Introduction

A whole range of families of graphs have been proposed for use as interconnection networks in the design of distributed-memory multiprocessors, where the vertices of a graph represent the processors of a machine and the edges between vertices the physical interconnections between processors. There are numerous properties such a family of graphs should have in order to be deemed suitable for such a purpose. For example, the graphs of such families should have small diameter (so as to aid message latency), be recursively decomposable (so as to aid scalability), have low degree (so as to lessen communication overheads), have high connectivity (so as to aid fault tolerance or data transfer), possess embeddings of other standard graphs (so as to aid simulations) and so on. The study of families of graphs suitable for use as interconnection networks has motivated new purely graph-theoretic research where the properties under consideration are relevant to the intended usage of these graphs in the context of parallel computation. Unfortunately the properties one might require are so diverse that there does not exist a family of graphs possessing every one of these properties and in practice trade-offs have to be made. Perhaps the most ubiquitous family of graphs in the landscape of interconnection networks are the hypercube Q_n and its close relation the k-ary n-cube Q_n^k (of course, such graphs are also common-place across discrete mathematics in general). The reader is referred to, for example, [6] for more on the relationship between graph theory and interconnection networks.

The study of Hamiltonian cycles in graphs is widespread. The basic problem of deciding whether an arbitrary graph has a Hamiltonian cycle is one of the canonical **NP**-complete problems, and there has been much research into restrictions upon arbitrary graphs under which this problem becomes solvable in polynomial-time. With respect to graphs used as interconnection networks, if such a graph possesses a Hamiltonian cycle then this cycle can, for example, easily be utilized so that all-to-all broadcasts can be accomplished and ring-based simulations undertaken (in the underlying distributed-memory multiprocessor). Many of the families of graphs used as interconnection networks possess Hamiltonian cycles: for instance, it has long been known that hypercubes [7] and k-ary n-cubes do [2].

Whilst the question of existence of Hamiltonian cycles in graphs such as hypercubes and k-ary n-cubes becomes a non-event, the question of efficiently constructing Hamiltonian cycles (especially using distributed-memory multiprocessors whose interconnection network is the graph in question) is still pertinent as is the consideration of additional impositions under which Hamiltonian cycles still exist. These impositions are usually related to avoiding or prescribing specific edges. For example, Chan and Lee [4] showed that an n-dimensional hypercube with at most $2n-5$ 'faulty links' (that is, with at most $2n-5$ edges missing) but where every vertex has degree at least 2, still has a Hamiltonian cycle but that there exist such faulty hypercubes with $2n-4$ faulty links (but so that every vertex has degree at least 2) that do not possess a Hamiltonian cycle. Ashir and Stewart [1] proved an analogous result for k-ary n-cubes with at most $4n-5$ faulty links. Chan and Lee also showed that it is **NP**-complete to decide whether a hypercube with an arbitrary set of faulty links is Hamiltonian [4], with Ashir and Stewart doing likewise for k-ary n-cubes [1].

However, it is with the prescription of specific vertices that we are concerned in this paper, which is, in some sense, complementary to edge avoidance. With regard to hypercubes, Caha and Koubek [3] proved that if the dimension n of a hypercube is at least 3 then for any set P of at most $n-1$ edges, there is a Hamiltonian cycle on which every edge of P lies if, and only if, the subgraph of the hypercube induced by the edges of P is a vertex-disjoint collection of paths. This result was extended by Dvořák [5] who showed that it holds when P consists of $2n-3$ edges and that this result is optimal (in that there are sets of $2n-2$ edges in an n-dimensional hypercube where the subgraph induced by these edges consists of vertex-disjoint paths but where there does not exist a Hamiltonian cycle upon which all of these edges lie). Consequently, Dvořák's result provides a precise classification as to when prescribed edges are guaranteed to lie upon a Hamiltonian cycle in a hypercube. Recently, Wang, Li and Wang [9] have embarked upon classifying when prescribed edges are guaranteed to lie upon a Hamiltonian cycle in a k-ary n-cube. Their result states that if P is a set of at most $2n-1$ edges in a 3-ary n-cube, where $n \geq 2$, then there is a Hamiltonian cycle on which every edge of P lies if, and only if, the subgraph of the 3-ary n-cube induced by the edges of P is a vertex-disjoint collection of paths. They make no comment as regards whether the number of edges in P can be increased so that the statement still holds (though they show that their result is optimal for $n = 2$) and pose the question of what happens in k-ary n-cubes when $k \geq 4$ as an open problem which we answer in this paper. In particular, we prove here that if P is a set of at most $2n-1$ edges in a k-ary n-cube, where

$n \geq 2$ and $k \geq 4$, then there is a Hamiltonian cycle on which every edge of P lies if, and only if, the subgraph of the k-ary n-cube induced by the edges of P is a vertex-disjoint collection of paths.

2 Basic Definitions and Results

For $n \geq 1$ and $k \geq 3$, the *k-ary n-cube* Q_n^k is the graph whose vertex set is $\{(u_n, u_{n-1}, \ldots, u_1) : u_i \in \{0, 1, \ldots, k-1\}$, for $i \in \{1, 2, \ldots, n\}\}$ and whose edge set consists of those pairs $((u_n, u_{n-1}, \ldots, u_1), (v_n, v_{n-1}, \ldots, v_1))$ where there exists some $d \in \{1, 2, \ldots, n\}$ such that $u_i = v_i$, whenever $i \neq d$, and either $u_d = v_d + 1$ or $u_d = v_d - 1$, with addition and subtraction modulo k (throughout this paper we assume that addition and subtraction on the components of vertices of Q_n^k is always modulo k). A path in some graph is a sequence of distinct vertices written $(x_1, x_2, \ldots, x_m)$, for some $m \geq 1$, so that (x_i, x_{i+1}) is an edge, for $i \in \{1, 2, \ldots, m-1\}$. The vertices x_1 and x_m of the path $(x_1, x_2, \ldots, x_m)$ are its *terminal vertices*, with all other vertices (when $m \geq 3$) its *internal vertices*. A path in a graph is *maximal* if it cannot be extended to a longer path in the graph. A cycle is a path $(x_1, x_2, \ldots, x_m)$, for some $m \geq 3$, for which we also have that (x_m, x_1) is an edge. Although a path and the corresponding cycle are written identically as sequences of vertices, it is always apparent as to whether we are referring to the sequence as a path or as a cycle.

Consider Q_n^k, where $n \geq 2$. Fix some $d \in \{1, 2, \ldots, n\}$. For any $i \in \{0, 1, \ldots, k-1\}$, consider those vertices of Q_n^k whose dth component is fixed at i. It is trivial to see that the subgraph of Q_n^k induced by these vertices is isomorphic to Q_{n-1}^k. We denote this subgraph by Q_i (when n, k and d are understood). We say that $Q_0, Q_1, \ldots, Q_{k-1}$ are formed by *partitioning Q_n^k over dimension d*. Note that any vertex x of Q_i has a corresponding vertex, denoted $n_j(x)$, in Q_j, for $j \in \{0, 1, \ldots, k-1\}$, where $n_j(x)$ is identical to x as a k-bit-string except that the dth component is equal to j. The vertex $x = n_i(x)$ is a neighbour of $n_{i-1}(x)$ and $n_{i+1}(x)$ in Q_n^k (with addition and subtraction on the indices modulo k), and the subgraph induced by the vertices of $\{x\} \cup \{n_j(x) : j \in \{0, 1, \ldots, k-1\} \setminus \{i\}\}$ is the cycle $(n_0(x), n_1(x), \ldots, n_{i-1}(x), x, n_{i+1}(x), \ldots, n_{k-1}(x))$. Any edge of Q_n^k that is not in $Q_0, Q_1, \ldots, Q_{k-1}$ is said to *lie in dimension d*. Let G be some subgraph of Q_i and let G' be the subgraph of Q_j, where $j \neq i$, induced by the edges of $\{(x, y) : (x, y)$ is an edge of Q_j such that $(n_i(x), n_i(y))$ is an edge of $G\}$. The graph G' is clearly isomorphic to G and is said to be the *isomorphic copy of G in Q_j*. Let X be a set of edges of Q_n^k. We write $\langle X \rangle$ to denote the subgraph induced by the edges of X. If every edge of X lies in some Q_i then the *isomorphic copy of X in Q_j*, where $i \neq j$, is the set of edges X' so that $\langle X' \rangle$ is the isomorphic copy of $\langle X \rangle$ in Q_j. We shall be interested in specific sets of edges in Q_n^k which we denote by P. We write P_j to denote those edges of Q_j that are in P, for $j \in \{0, 1, \ldots, k-1\}$. Edges in $P \setminus \cup_{i=0}^{k-1} P_i$ clearly lie in dimension d.

Suppose that we have partitioned Q_n^k over some dimension d to get $Q_0, Q_1, \ldots, Q_{k-1}$ and we have a specific set of edges P. If (x,y) is an edge of Q_i then we say that the cycle $(x, n_{i+1}(x), n_{i+1}(y), y)$ is a *bridge joining* Q_i *and* Q_{i+1}. Of course, $(n_{i+1}(x), n_i(n_{i+1}(x)), n_i(n_{i+1}(y)), n_{i+1}(y)) = (n_{i+1}(x), x, y, n_{i+1}(y))$ is the same bridge. Let $(x, n_{i+1}(x), n_{i+1}(y), y)$ be a bridge joining Q_i and Q_{i+1}. We say that this bridge is *right-useable* if

1. $(n_{i+1}(x), n_{i+1}(y)) \notin P_{i+1}$
2. $n_{i+1}(x)$ and $n_{i+1}(y)$ are not terminal vertices on some maximal path of $\langle P_{i+1} \rangle$ of length at least 2
3. both $n_{i+1}(x)$ and $n_{i+1}(y)$ are incident with at most 1 edge of P_{i+1}.

Note that if $\langle P_{i+1} \rangle$ consists of a set of vertex-disjoint paths and the bridge $(x, n_{i+1}(x), n_{i+1}(y), y)$ is right-useable then by conditions 2 and 3 of the definition of right-useability, $\langle P_{i+1} \cup \{(n_{i+1}(x), n_{i+1}(y))\} \rangle$ consists of a set of vertex-disjoint paths also. Our bridge is *left-useable* if $(x,y) \notin P_i$, x and y are not terminal vertices on some maximal path of $\langle P_i \rangle$ of length at least 2, and both x and y are incident with at most 1 edge of P_i. Our bridge is *useable* if it is both left-useable and right-useable. We shall use bridges to build larger cycles out of smaller cycles as follows. Suppose that C_i is a cycle in Q_i and C_{i+1} is a cycle in Q_{i+1} so that (x,y) is an edge of C_i and $(n_{i+1}(x), n_{i+1}(y))$ is an edge of C_{i+1}. The cycle D formed by removing the edges (x,y) and $(n_{i+1}(x), n_{i+1}(y))$ from C_i and C_{i+1}, respectively, and including the edges $(x, n_{i+1}(x))$ and $(y, n_{i+1}(y))$ is said to have been formed by *joining* C_i *and* C_{i+1} *using the bridge* $(x, n_{i+1}(x), n_{i+1}(y), y)$. We also say that D has been formed by *extending* C_i or C_{i+1}.

The following result will prove very useful.

Theorem 1. *[8] Let $k \geq 3$ be odd and let $n \geq 2$. Given any two distinct vertices x and y of Q_n^k, there exists a Hamiltonian path from x to y.*

Our primary motivation is the following result due to Wang, Li and Wang.

Theorem 2. *[9] Let $n \geq 2$ and let P be any set of $2n-1$ edges in Q_n^3. The 3-ary n-cube Q_n^3 has a Hamiltonian cycle on which every edge of P lies if, and only if, $\langle P \rangle$ consists of pairwise vertex-disjoint paths.*

3 The Main Result

In this section we prove our main result, namely Theorem 4. The proof of Theorem 4 is by induction. Due to space limitations, we simply state the base case as Theorem 3 (though it is not difficult to prove).

Theorem 3. *Let $k \geq 4$ and let P be a set of 3 edges in Q_2^k. The k-ary 2-cube Q_2^k has a Hamiltonian cycle on which every edge of P lies if, and only if, $\langle P \rangle$ consists of pairwise vertex-disjoint paths.*

Theorem 4. *Let $n \geq 2$, let $k \geq 4$ and let P be a set of edges of Q_n^k with $|P| \leq 2n-1$. There exists a Hamiltonian cycle of Q_n^k on which every edge of P lies if, and only if, $\langle P \rangle$ consists of pairwise vertex-disjoint paths.*

Proof. Note that because $k \geq 4$, we may assume that Q_n^k contains no cycles of length 3. We proceed by induction on n. As our induction hypothesis, suppose that $n \geq 3$ and that whenever we have a set P' of at most $2n-3$ edges of Q_{n-1}^k, there exists a Hamiltonian cycle of Q_{n-1}^k on which every edge of P' lies if, and only if, $\langle P' \rangle$ consists of pairwise vertex-disjoint paths. The base case of our induction follows by Theorem 3. Let P be a set of at most $2n-1$ edges in Q_n^k. If Q_n^k has a Hamiltonian cycle containing every edge of P then trivially $\langle P \rangle$ consists of pairwise vertex-disjoint paths. So, assume that $\langle P \rangle$ consists of pairwise vertex-disjoint paths.

There exists some dimension d such that at most 1 edge of P lies in dimension d. Partition Q_n^k over dimension d to obtain the k-ary $(n-1)$-cubes $Q_0, Q_1, \ldots, Q_{n-1}$, with P_j consisting of those edges of P that lie in Q_j, for $j \in \{0, 1, \ldots, k-1\}$. W.l.o.g. assume that $|P_0| \geq |P_j|$, for $j \in \{1, 2, \ldots, k-1\}$. There are two essential cases: when there is 1 edge in dimension d; and when there are no edges in dimension d. However, we begin with two useful lemmas.

Lemma 1. *Let X be a set of edges in Q_i where $\langle X \rangle$ consists of a set of vertex-disjoint paths or cycles. Let the set of edges X' be the set of edges of Q_{i+1} isomorphic to X. The number of right-useable bridges of the form $(x, n_{i+1}(x), n_{i+1}(y), y)$ joining Q_i and Q_{i+1}, where $(x, y) \in X$, is at least* $\max\{|X| - |X' \cap P_{i+1}| - 2|P_{i+1}|, 0\}$.

Proof. Throughout this proof, by a bridge we mean a bridge of the form $(x, n_{i+1}(x), n_{i+1}(y), y)$, where $(x, y) \in X$. Consider some edge $f \in P_{i+1}$. If $f \in X'$ then this makes the bridge containing f not right-useable: so, $|X' \cap P_{i+1}|$ bridges are not right-useable because condition 1 of the definition of right-useability fails.

Suppose that the edge $f \in X' \setminus P_{i+1}$ is such that its 2 incident vertices are the terminal vertices of a maximal path of $\langle P_{i+1} \rangle$ of length at least 2. As there are no cycles of length 3, this path must have length at least 3. Thus, if α is the number of vertex-disjoint maximal paths in $\langle P_{i+1} \rangle$ of length at least 3 then at most α bridges are not right-useable because condition 2 of the definition of right-useability fails.

Consider some edge $f \in X' \setminus P_{i+1}$ where its 2 incident vertices are not the terminal vertices of a maximal path of $\langle P_{i+1} \rangle$ of length at least 2. However, suppose that the bridge involving f is still not right-useable. So, one of its incident vertices is incident with 2 edges of P_{i+1}: that is, this vertex is an internal vertex of some maximal path in $\langle P_{i+1} \rangle$. As $\langle X' \rangle$ is such that every vertex has degree at most 2, any such internal vertex renders at most 2 bridges not right-useable. Hence, the maximum number of bridges rendered not right-useable because condition 3 of the definition of right-useability fails is at most $2(|P_{i+1}| - \beta)$, where β is the number of vertex-disjoint maximal paths in $\langle P_{i+1} \rangle$.

Consequently, the total number of bridges rendered not right-useable is at most $|X' \cap P_{i+1}| + \alpha + 2(|P_{i+1}| - \beta) \leq |X' \cap P_{i+1}| + 2|P_{i+1}|$. □

Lemma 2. *Let D be a cycle spanning all vertices of $Q_i, Q_{i+1}, \ldots, Q_l$, for some i and l (with possibly $i = l$), where*

– $|P_j| \leq 2n-4$, *for* $j \in \{0, 1, \ldots, k-1\} \setminus \{i, i+1, \ldots, l\}$

– *D contains all edges of $P_i \cup P_{i+1} \cup \ldots \cup P_l$ as well as any dimension d edge of P that might happen to join vertices of $Q_i, Q_{i+1}, \ldots, Q_l$*
– *there are no edges of P lying in dimension d and incident with a vertex from Q_j, for $j \in \{0, 1, \ldots, k-1\} \setminus \{i, i+1, \ldots, l\}$*
– *the number of edges of D lying in Q_i is greater than $6n - 9$.*

The cycle D can be extended to a Hamiltonian cycle of Q_n^k containing every edge of P.

Proof. Let X be the set of edges of D lying in Q_i and let X' be the isomorphic copy of X in Q_{i-1}. By Lemma 1, there are at least $\max\{|X| - |X' \cap P_{i-1}| - 2|P_{i-1}|, 0\}$ left-useable bridges joining D and Q_{i-1}. Moreover, at least $\max\{|X| - |X' \cap P_{i-1}| - |X \cap P_i| - 2|P_{i-1}|, 0\} = \max\{|X| - (2n-1) - 2(2n-4), 0\} = \max\{|X| - 6n + 9, 0\} > 0$ of these bridges are such that the edge of the bridge lying in Q_i is not in P_i.

Let (u, u', v', v) be such a left-useable bridge joining the edge (u, v) of D lying in Q_i to the edge (u', v') of Q_{i-1}. By the induction hypothesis applied to $(P_{i-1} \cup \{(u', v')\}, Q_{i-1})$ (see the remark immediately after the definition of right-useability), there exists a Hamiltonian cycle C_{i-1} in Q_{i-1} containing every edge of P_{i-1} as well as the edge (u', v'). Join D and C_{i-1} using the bridge (u, u', v', v) to obtain a cycle spanning the vertices of $Q_{i-1}, Q_i, \ldots, Q_l$ and containing every edge of $P_{i-1} \cup P_i \cup \ldots \cup P_l$ as well as any dimension d edge of P that might happen to join vertices of $Q_i, Q_{i+1}, \ldots, Q_l$. Proceeding iteratively in this way (noting that $k^{n-1} - 1 > 6n - 9$ and repeatedly applying Lemma 1) yields the result. □

<u>Case (a)</u>: $|P \setminus \cup_{i=0}^{k-1} P_i| = 1$.

Let the edge of P that is not in $\cup_{i=0}^{k-1} P_i$ be $e = (x, y)$.

Lemma 3. *Suppose that x lies in Q_i, y lies in Q_{i+1} and both $|P_i|$ and $|P_{i+1}|$ are at most $2n - 4$. There exists a cycle D spanning all vertices of Q_i and Q_{i+1} that contains every edge of $P_i \cup P_{i+1} \cup \{e\}$ and is such that only 2 edges of D do not lie in Q_i or Q_{i+1} (one of which is e) with these 2 edges being part of a bridge joining Q_i and Q_{i+1}.*

Proof. Let x' be a neighbour of x in Q_i with y' the corresponding neighbour of y in Q_{i+1} (so $y' = n_{i+1}(x')$). Consider the bridge (x, x', y', y). Suppose that this bridge is useable. We can apply the induction hypothesis to $(P_i \cup \{(x, x')\}, Q_i)$ and to $(P_{i+1} \cup \{(y, y')\}, Q_{i+1})$ and obtain Hamiltonian cycles C_i and C_{i+1} of Q_i and Q_{i+1}, respectively, so that C_i contains every edge of P_i, as well as (x, x'), and C_{i+1} contains every edge of P_{i+1}, as well as (y, y'). We can join C_i and C_{i+1} using the bridge (x, x', y', y) to obtain a cycle D as in the lemma.

Suppose that the bridge (x, x', y', y) is not useable. So, as x is incident with at most 1 edge of P_i and at most 1 edge of P_{i+1}, we must have (at least) one of the following six occurrences: $(x, x') \in P_i$; x and x' are the terminal vertices of some maximal path in $\langle P_i \rangle$ of length at least 2; x' is incident with at least 2 edges of P_i; $(y, y') \in P_{i+1}$; y and y' are the terminal vertices of some maximal path in $\langle P_{i+1} \rangle$; y' is incident with at least 2 edges of P_{i+1}. Let us count the maximal number of bridges joining Q_i and Q_{i+1} involving x that are rendered not useable due to the edges of P_i, *i.e.*, not left-useable. As x is incident with at most 1 edge of P_i, at most 1 bridge of the form (x, x'', y'', y) is rendered not useable because $(x, x'') \in P_i$ or because x and x'' are the terminal vertices of some maximal path in $\langle P_i \rangle$ of length at least 2. Alternatively, if a bridge of the form (x, x', y', y) is rendered not useable because x' is incident with at least 2 edges of P_i then x' must be some internal vertex of a maximal path of $\langle P_i \rangle$.

– Suppose that there exists an edge $(x, x'') \in P_i$. This leaves at most $|P_i| - 2$ internal vertices of paths in $\langle P_i \rangle$ that might be used to render bridges involving x not useable.
– Suppose that x and x'' are the terminal vertices of some maximal path in $\langle P_i \rangle$ of length at least 2. Consider the vertex z of this path adjacent to x''. The vertex z cannot be adjacent to x as Q_n^k has no cycles of length 3. Thus, the (internal) vertex z (of our path) cannot be used to render a bridge involving x not useable, and this leaves at most $|P_i| - 2$ internal vertices of maximal paths in $\langle P_i \rangle$ that can.
– Suppose that there does not exist an edge $(x, x'') \in P_i$ nor is it the case that there exists a neighbour x'' of x in Q_i such that x and x'' are the terminal vertices of some maximal path in $\langle P_i \rangle$ of length at least 2. The maximal number of internal vertices of paths in $\langle P_i \rangle$ that can be used to render a bridge involving x not useable is at most $|P_i| - 1$.

So, the maximal number of bridges involving x not useable because of edges of P_i is at most $|P_i| - 1$. The same goes for the edges of P_{i+1}. Thus, as: there are $2n - 2$ bridges involving x and at most $|P_i| + |P_{i+1}| - 2$ are not useable; and $|P_i| + |P_{i+1}| \leq 2n - 1$, at least one bridge must be useable. The result follows. □

Suppose that $|P_0| \leq 2n - 4$. By Lemmas 2 and 3, there is a Hamiltonian cycle in Q_n^k containing all edges of P. So, we assume that $|P_0| \geq 2n - 3$.

Lemma 4. *Suppose that x is a vertex of Q_0, y is a vertex of Q_1 and $|P_0| = 2n - 3$. There is a Hamiltonian cycle in Q_n^k containing all edges of P.*

Proof. By the induction hypothesis applied to (P_0, Q_0), there is a Hamiltonian cycle in Q_0 containing all edges of P_0. Let x' and x'' be the neighbours of x on C_0, with y' and y'', respectively, their neighbours in Q_1. W.l.o.g. $(x, x') \notin P_0$ (as $(x, y) \in P$). Consider the bridge (x, y, y', x'). If $(y, y') \notin P_1$ then apply the induction hypothesis to $(P_1 \cup \{(y, y')\}, Q_1)$ to obtain a Hamiltonian cycle C_1 in Q_1 that contains all edges of P_1 as well as (y, y'). We can join C_0 and C_1 using the bridge (x, y, y', x') to obtain a cycle spanning all vertices of Q_0 and Q_1 and containing all edges of $P_0 \cup P_1 \cup \{e\}$. The result follows by Lemma 2.

So, suppose that $(y, y') \in P_1$: that is, $P_1 = \{(y, y')\}$ and $\cup_{j=2}^{k-1} P_j = \emptyset$. Let C_1' be the isomorphic copy of C_0 in Q_1. In particular, C_1' contains (y, y'). Consider the path obtained by starting from y'' and traversing C_1' to y (omitting (y'', y)), taking the edge (y, x), and then traversing C_0 to x' (omitting (x, x')). This path contains every vertex of Q_0 and Q_1 and every edge of P. Extend this path by the edges $(y'', n_2(x''))$ and $(x', n_{k-1}(x'))$ to obtain the path ρ.

If $k = 5$ then we can, first, apply Theorem 1 three times to obtain Hamiltonian paths in Q_2, Q_3 and Q_4 from $n_2(x'')$ to $n_2(x')$, from $n_3(x'')$ to $n_3(x')$, and from $n_4(x'')$ to $n_4(x')$, respectively, and, second, easily compose these paths with ρ to obtain a Hamiltonian cycle in Q_n^5 containing all edges of P. Indeed, by proceeding similarly, we can obtain a Hamiltonian cycle in any Q_n^k containing all edges of P whenever $k \geq 5$ is odd.

Suppose that $k = 4$. Applying the induction hypothesis to $(\{(n_2(x''), n_2(x))\}, Q_2)$ and to $(\{(n_3(x'), n_3(x))\}, Q_3)$ yields Hamiltonian cycles C_2 and C_3 in Q_2 and Q_3, respectively: that is, Hamiltonian paths in Q_2 from $n_2(x'')$ to $n_2(x)$ and in Q_3 from $n_3(x')$ to $n_3(x)$. These paths can easily be composed with ρ to obtain a Hamiltonian cycle in Q_n^4 containing all edges of P. When $k \geq 6$ is even, we proceed similarly by applying the induction hypothesis to $(\{(n_2(x''), n_2(x))\}, Q_2)$ and to $(\{(n_j(x'), n_j(x))\}, Q_j)$, if $3 \leq j \leq k-1$, before composing the resulting paths. The result follows. □

Lemma 5. *Suppose that x is a vertex of Q_i, y is a vertex of Q_{i+1} and $|P_0| = 2n-3$, where $i \neq 0 \neq i+1$. There is a Hamiltonian cycle in Q_n^k containing all edges of P.*

Proof. As in the proof of Lemma 3, there is a useable bridge (x, y, y', x') joining Q_i and Q_{i+1}. Let the edge of $P \setminus (P_0 \cup \{e\})$ be (p, q). By the induction hypothesis applied to (P_0, Q_0), there is a Hamiltonian cycle C_0 in Q_0 containing every edge of P_0. The cycle C_0 contains $k^{n-1} - (2n-3) \geq 13$ edges not in P_0.

Choose two non-incident edges (u, v) and (s, t) of C_0 that are not in P_0 where $\{u, v, s, t\} \cap \{n_0(x), n_0(x'), n_0(p), n_0(q)\} = \emptyset$. Applying the induction hypothesis to $(P_i \cup \{(x, x'), (n_i(u), n_i(v))\}, Q_i)$ yields a Hamiltonian cycle in Q_i containing all edges of P_i as well as (x, x') and $(n_i(u), n_i(v))$. Applying the induction hypothesis to $(P_j \cup \{(n_j(u), n_j(v)), (n_j(s), n_j(t))\}, Q_j)$, for $j \in \{2, 3, \ldots, i-1\}$, yields a Hamiltonian cycle C_j in Q_j containing all edges of P_j as well as $(n_j(u), n_j(v))$ and $(n_j(s), n_j(t))$. We can join C_i and C_{i-1} using the bridge $(n_i(u), n_{i-1}(u), n_{i-1}(v), n_i(v))$, and then using the bridge $(n_j(u), n_{j-1}(u), n_{j-1}(v), n_j(v))$ or $(n_j(s), n_{j-1}(t), n_{j-1}(t), n_j(s))$, for $j \in \{1, 2, \ldots, i-1\}$, we can join the Hamiltonian cycles $C_0, C_1, \ldots, C_i$ so as to obtain a cycle D spanning all vertices of $Q_0, Q_1, \ldots, Q_i$ and containing all edges of $P_0 \cup P_1 \cup \ldots \cup P_i$.

Applying the induction hypothesis to $(P_{i+1} \cup \{(y, y')\}, Q_{i+1})$ yields a Hamiltonian cycle in Q_{i+1} containing all edges of P_{i+1} and (y, y'). Join D and C_{i+1} using the bridge (x, y, y', x') to obtain a cycle spanning all vertices of $Q_0, Q_1, \ldots, Q_{i+1}$ and containing all edges of $P_0 \cup P_1 \cup \ldots \cup P_{i+1} \cup \{e\}$. This cycle can be extended to a Hamiltonian cycle in Q_n^k containing all edges in P by Lemma 2. □

Suppose that $|P_0| = 2n-3$. By Lemmas 4 and 5, there is a Hamiltonian cycle in Q_n^k containing all edges of P. Henceforth, we assume that $|P_0| = 2n-2$.

There are two possibilities: x is a vertex of Q_0 and y is a vertex of Q_1; x is a vertex of Q_i, where $i \neq 0$, and y is a vertex of Q_{i+1}, where $i+1 \neq 0$.

Lemma 6. *Suppose that x is a vertex of Q_0, y is a vertex of Q_1 and $|P_0| = 2n-2$. There is a Hamiltonian cycle in Q_n^k containing every edge of P.*

Proof. As x is incident with at most 1 edge of P_0, let (a,b) be some edge of P_0 that is not incident with x. Applying the induction hypothesis to $(P_0 \setminus \{(a,b)\}, Q_0)$ results in a Hamiltonian cycle C_0 in Q_0 containing every edge of $P_0 \setminus \{(a,b)\}$. Suppose that C_0 also contains (a,b). Let x' and x'' be the neighbours of x on C_0, with y' and y'', respectively, their neighbours in Q_1. W.l.o.g. $(x,x') \notin P_0$ (as $(x,y) \in P$). Consider the bridge (x,y,y',x'). Apply the induction hypothesis to $(\{(y,y')\}, Q_1)$ to obtain a Hamiltonian cycle C_1 in Q_1 that contains (y,y'). We can join C_0 and C_1 using the bridge (x,y,y',x') to obtain a cycle spanning all vertices of Q_0 and Q_1 and containing all edges of P. The result follows by Lemma 2.

So, suppose that $(a,b) \notin P_0$. Let a' and a'' be the neighbours of a on C_0, and let b' and b'' be the neighbours of b on C_0. Moreover, assume that there is a sub-path of C_0 joining to a' to b' on which x does not lie. Let x' be the neighbour of x on C_0 so that x' does not lie on the sub-path of C_0 from x to a avoiding b, and let x'' be the other neighbour of x on C_0. W.l.o.g. we may assume that $(x,x') \notin P_0$ (as $(x,y) \in P$). Note that $a' \neq b'$ as otherwise there would be a cycle of length 3 in Q_n^k. However, it may be the case that $x' = b''$, $x' = b$, $x'' = a''$ or $x'' = a$.

There are 4 different essential cases to consider depending upon whether or not the edges (a,a') and (b,b') are in P_0. Some of these cases have sub-cases. All cases and their sub-cases are described below and illustrated in Fig. 1, as are the paths ρ_1 and ρ_2 (except for Cases 2.b, 2.c and 3.b where there is only ρ_1 and Case 4.a where there is also the path ρ_3).

1. $(a,a') \notin P_0$ and $(b,b') \notin P_0$.

2.a $(a,a') \notin P_0$, $(b,b') \in P_0$ and $b'' \neq x' \neq b$ (note that $(b,b'') \notin P_0$).

2.b $(a,a') \notin P_0$, $(b,b') \in P_0$ and $b'' = x'$ (note that the path ρ_2 does not include $x' = b''$ and that $(b,x') \notin P_0$).

2.c $(a,a') \notin P_0$, $(b,b') \in P_0$ and $x' = b$.

3.a $(a,a') \in P_0$, $(b,b') \notin P_0$ and $a'' \neq x$ (note that $(a,a'') \notin P_0$).

3.b $(a,a') \in P_0$, $(b,b') \notin P_0$ and $a'' = x$ (note that $(a,a'') \notin P_0$).

4.a If $(a,a') \in P_0$, $(b,b') \in P_0$, $x'' \neq a$ and $x' \neq b''$ then let $(c,d) \notin P_0$ be an edge on the sub-path of C_0 joining a' and b' avoiding a so that c is closer to a' on this path than d is and
   - define ρ_1 to be the sub-path of C_0 from c to a avoiding b, concatenated with (a,b), concatenated with the sub-path of C_0 from b to d avoiding a
   - define ρ_2 to be the sub-path of C_0 from x to a'' avoiding a
   - if $x' \neq b$ then define ρ_3 to be the sub-path of C_0 from x' to b'' avoiding a

   (note that both (a,a'') and (b,b'') are not in P_0).

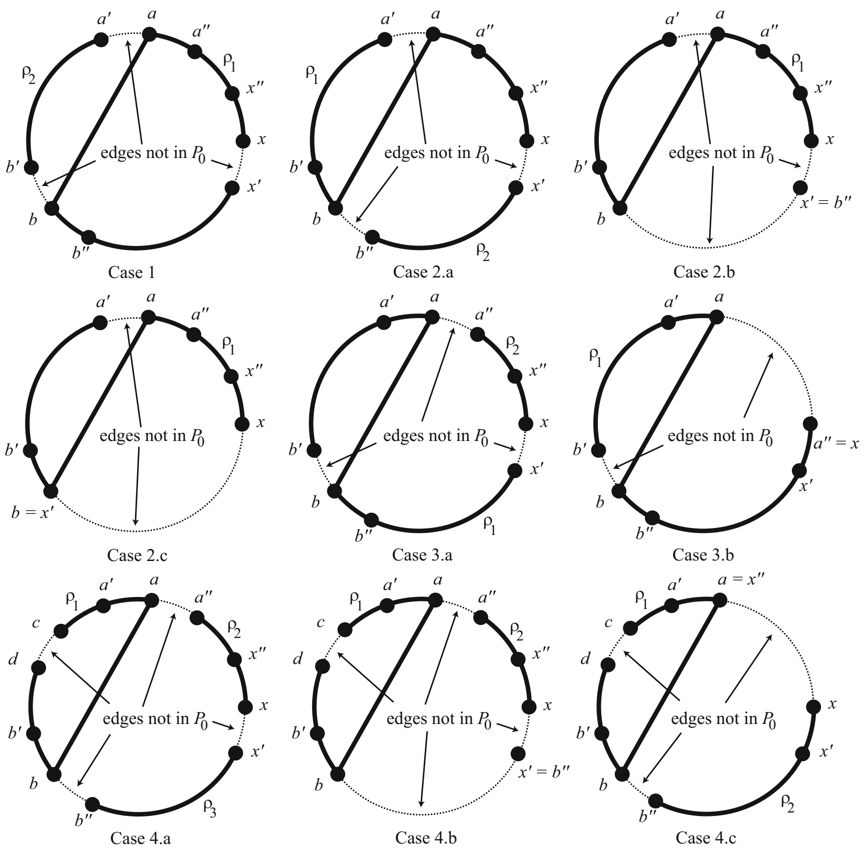


Fig. 1. The different cases for the edge (a, b)

4.b If $(a, a') \in P_0$, $(b, b') \in P_0$, $x'' \neq a$ and $x' = b''$ then let $(c, d) \notin P_0$ be an edge on the sub-path of C_0 joining a' and b' avoiding a so that c is closer to a' on this path than d is and

- define ρ_1 to be the sub-path of C_0 from c to a avoiding b, concatenated with (a, b), concatenated with the sub-path of C_0 from b to d avoiding a
- define ρ_2 to be the sub-path of C_0 from x to a'' avoiding a

(note that these paths do not include $x' = b''$ and that both (a, a'') and (b, b'') are not in P_0).

4.c If $(a, a') \in P_0$, $(b, b') \in P_0$ and $x'' = a$ then let $(c, d) \notin P_0$ be an edge on the sub-path of C_0 joining a' and b' avoiding a so that c is closer to a' on this path than d is and

- define ρ_1 to be the sub-path of C_0 from c to a avoiding b, concatenated with (a, b), concatenated with the sub-path of C_0 from b to d avoiding a
- define ρ_2 to be the sub-path of C_0 from x to b'' avoiding a

(note that both (a, a'') and (b, b'') are not in P_0, and that $x' \neq b$ as otherwise there would be a cycle of length 3 in Q_n^k).

Consider Case 1. Let ρ_1' and ρ_2' be the isomorphic copies of ρ_1 and ρ_2, respectively, in Q_1. Join ρ_1 and ρ_1' using the edges $(a', n_1(a'))$ and $(b', n_1(b'))$ to form the cycle D_1, and join ρ_2 and ρ_2' using the edges (x, y) and $(x', n_1(x'))$ to form the cycle D_2. Every edge of P lies on one of these cycles. Take any edge f of D_1 lying within Q_1 and any edge g of D_2 lying within Q_1, and let f' and g' be the isomorphic copies of f and g, respectively, in Q_2. By the induction hypothesis applied to $(\{f', g'\}, Q_2)$, there is a Hamiltonian cycle C_2 in Q_2 containing f' and g'. Join D_1 and D_2 to C_2 using the bridges involving f and f' and g and g', respectively. We obtain a cycle spanning all vertices of Q_0, Q_1 and Q_2 and containing all edges of P. We can extend this cycle to a Hamiltonian cycle of Q_n^k containing all edges of P by Lemma 2 (as $k^{n-1} - 2 > 6n - 9$). Analogous constructions apply in Cases 2.a, 2.c, 3.a, 3.b and 4.c.

We are left with Cases 2.b, 4.a and 4.b. Consider Case 2.b. Let ρ_1' be the isomorphic copy of ρ_1 in Q_1. Join ρ_1 and ρ_1' using the edges (x, y) and $(a', n_1(a'))$ to form the cycle D.

Suppose that k is even. For every $j \in \{2, 3, \ldots, k-1\}$, apply the induction hypothesis to $(\{(n_j(x'), n_j(x)), (n_j(x), n_j(x''))\}, Q_j)$ to obtain a Hamiltonian cycle C_j in Q_j upon which both edges $(n_j(x'), n_j(x))$ and $(n_j(x), n_j(x''))$ lie. For $j \in \{2, 3, \ldots, k-1\}$, let π_j by the sub-path of C_j from $n_j(x')$ to $n_j(x)$ of length $k^{n-1} - 1$. Form the cycle D' by starting from the path $(x', n_1(x'), n_2(x'))$, concatenating π_2, concatenating the edge $(n_2(x), n_3(x))$, concatenating the path π_3, concatenating the edge $(n_3(x'), n_4(x'))$, concatenating π_4, …, concatenating the edge $(n_{k-2}(x), n_{k-1}(x))$, concatenating the path π_{k-1} and finally concatenating the edge $(n_{k-1}(x'), x')$. The cycles D and D' span all vertices of Q_n^k. Join D and D' using the bridge $(y, n_2(x), n_2(x''), n_1(x''))$ to obtain a Hamiltonian cycle of Q_n^k containing all edges of P (note that neither $(y, n_1(x''))$ nor $(n_2(x), n_2(x''))$ lies in P). The construction can be visualised as in Fig. 2. There is an analogous construction for Case 4.b except that instead of one cycle D we have two cycles D_1 and D_2, formed by composing the paths ρ_1 and ρ_1' and the paths ρ_2 and ρ_2', respectively. We build the cycle D' as we did before except that when building D' we ensure that all Hamiltonian cycles C_j, for $j \in \{2, 3, \ldots, k-1\}$, contain the edges of $\{(n_j(x'), n_j(x)), (n_j(x), n_j(x'')), (n_j(a), n_j(b))\}$. We join D_1 and D using the bridge $(n_1(a), n_2(a), n_2(b), n_1(b))$ and the resulting cycle to D_2 using the bridge $(y, n_2(x), n_2(x''), n_1(x''))$.

Suppose that k is odd. Let ρ_1'' be the isomorphic copy of ρ_1 in Q_2. For every $j \in \{3, 4, \ldots, k-2\}$, we use Lemma 1 to obtain a Hamiltonian path π_j in Q_j from $n_j(a')$ to $n_j(x)$. We use Lemma 1 to obtain a Hamiltonian path π_{k-1} in Q_{k-1} from $n_{k-1}(x)$ to $n_{k-1}(x')$. We build the cycle D' by starting from the path $(x', n_1(x'), n_2(x'))$, concatenating the edge $(n_2(x'), n_2(x))$, concatenating the path ρ_1'', concatenating the edge $(n_2(a'), n_3(a'))$, concatenating the path π_3, concatenating the edge $(n_3(x), n_4(x))$, concatenating the path π_4, concatenating the edge $(n_4(a'), n_5(a'))$, concatenating the path π_5, …, concatenating the edge $(n_{k-2}(x), n_{k-1}(x))$, concatenating the path π_{k-1} and concatenating the edge $(n_{k-1}(x'), x')$. The cycles D and D' span all vertices of Q_n^k. Join D and D' using the bridge $(y, n_2(x), n_2(x''), n_1(x''))$ to obtain a Hamiltonian cycle of Q_n^k

containing all edges of P (note that neither $(y, n_1(x''))$ nor $(n_2(x), n_2(x''))$ lies in P). There is an analogous construction for Case 4.b except that instead of one cycle D we have two cycles D_1 and D_2, formed by composing the paths ρ_1 and ρ_1' and the paths ρ_2 and ρ_2', respectively. We build the cycle D' as we did before except that when building D' we ensure that: the Hamiltonian path π_2 in Q_2 is the sub-path of length $k^{n-1} - 1$ of the isomorphic copy of C_0 in Q_2 from $n_2(x')$ to $n_2(b)$; the Hamiltonian paths π_j in Q_j are from $n_j(b)$ to $n_j(x)$, for $j \in \{3, 4, \ldots, k-2\}$; and the Hamiltonian path π_{k-1} in Q_{k-1} is from $n_{k-1}(x)$ to $n_{k-1}(x')$. We join D_1 and D' using the bridge $(n_1(a), n_2(a), n_2(a'), n_1(a'))$, and we join the resulting cycle with D_2 using the bridge $(y, n_2(x), n_2(x''), n_1(x''))$ to obtain a Hamiltonian cycle of Q_n^k containing all edges of P (see Fig. 3).

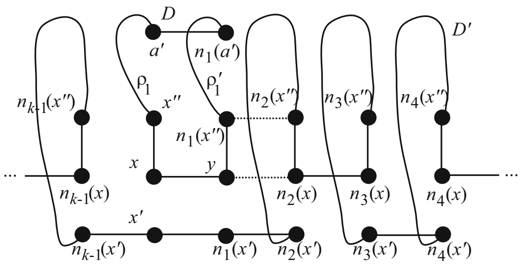


Fig. 2. Case 2.b when k is even

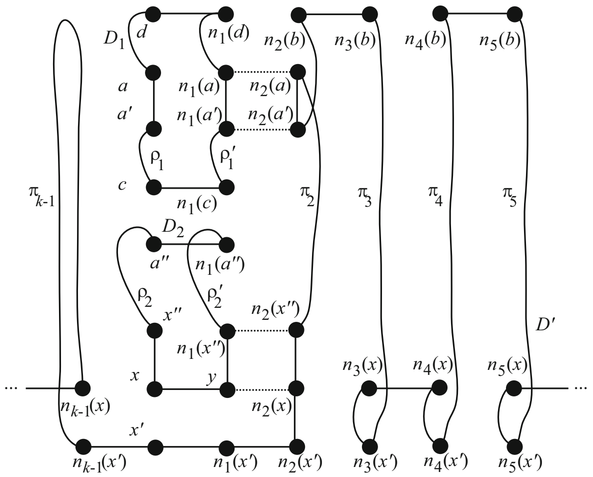


Fig. 3. Case 4.b when k is odd

Finally, consider Case 4.a. Let ρ_1', ρ_2' and ρ_3' be the isomorphic copies of ρ_1, ρ_2 and ρ_3, respectively, in Q_1. Join corresponding pairs to form three cycles D_1, D_2 and D_3, respectively, which span all vertices of Q_0 and Q_1. Choose an edge f_i of D_i that lies in Q_1 and let f_i' be the isomorphic copy in Q_2, for $i = 1, 2, 3$. Apply the induction hypothesis to $(\{f_1', f_2', f_3'\}, Q_2)$ to obtain a Hamiltonian cycle D in Q_2 containing f_1', f_2' and f_3'. Join D to D_1, D_2 and D_3 using the corresponding bridge to obtain a cycle D' spanning all vertices of Q_1, Q_2 and Q_3 and containing all edges of P. The result follows by Lemma 2. $\square$

Lemma 7. *Suppose that x is a vertex of Q_i and y is a vertex of Q_{i+1} where $i \neq 0 \neq i+1$. Suppose that $|P_0| = 2n-2$. There is a Hamiltonian cycle in Q_n^k containing every edge of P.*

Proof. Let (p,q) be some edge of P_0. By the induction hypothesis applied to $(P \setminus \{(p,q)\}, Q_0)$, there is a Hamiltonian cycle C_0 in Q_0 containing every edge of $P_0 \setminus \{(p,q)\}$. If (p,q) lies in C_0 then let D be the cycle C_0. Suppose that (p,q) does not lie on C_0. There are two possibilities: we have a Hamiltonian path ρ_1 in Q_0 (a sub-path of C_0) containing all edges of P_0; or we have two vertex-disjoint (non-trivial) paths ρ_1 and ρ_2 in Q_0 (sub-paths of C_0) which span all vertices of Q_0 and contain all edges of P_0 (see the diagrams in Case 1 and Case 2.a in Fig. 1). In the first case, let ρ_1' be the isomorphic copy of ρ_1 in Q_1 and let D be the cycle spanning all vertices of Q_0 and Q_1 obtained by joining ρ_1 and ρ_1'. In the second case, let ρ_1' and ρ_2' be the isomorphic copies of ρ_1 and ρ_2, respectively, in Q_1, and let D_1 and D_2 be the cycles obtained by joining ρ_1 and ρ_1' and by joining ρ_2 and ρ_2', respectively, so that the cycles D_1 and D_2 span the vertices of Q_0 and Q_1. Now choose some edge f_1 of D_1 that lies in Q_1 and some edge f_2 of D_2 that lies in Q_1, ensuring that f_1 is incident with x if x lies in Q_1. Let f_1' and f_2' be the isomorphic copies of f_1 and f_2, respectively, in Q_2. By the induction hypothesis applied to $(\{f_1', f_2'\}, Q_2)$, there is a Hamiltonian cycle C_2 in Q_2 containing f_1' and f_2'. Join D_1 and D_2 to C_2 using the bridges involving f_1 and f_1' and f_2 and f_2' to obtain a cycle D.

So, we have a cycle D containing every edge of P_0. We iteratively work through the remaining k-ary $(n-1)$-cubes not yet spanned by the cycle D and, using the induction hypothesis, extend D so that the edge (x,y) appears in the extension of D (we do this as we did above by always choosing the bridge by which we extend so that it contains (x,y)). The result follows. $\square$

Suppose that $|P_0| = 2n-1$. By Lemmas 6 and 7, there is a Hamiltonian cycle in Q_n^k containing all edges of P.

<u>Case (b)</u>: $P = \cup_{i=0}^{k-1} P_i$.

Suppose that $|P_0| \leq 2n-3$. By the induction hypothesis applied to (P_0, Q_0), there is a Hamiltonian cycle C_0 in Q_0 containing every edge of P_0. The result follows by Lemma 2.

Suppose that $|P_0| = 2n-2$. Let (p,q) be some edge of P_0. By the induction hypothesis applied to $(P_0 \setminus \{(p,q)\}, Q_0)$, there is a Hamiltonian cycle C_0 of Q_0 containing every edge of $P_0 \setminus \{(p,q)\}$. If (p,q) lies on C_0 then set $D = C_0$. Suppose that (p,q) does not lie on C_0. There are two possibilities: we have a Hamiltonian path ρ_1 in Q_0 (a sub-path of C_0) containing all edges of P_0; or we have two vertex-disjoint (non-trivial) paths ρ_1 and ρ_2 in Q_0 (sub-paths of C_0) which span all vertices of Q_0 and contain all edges of P_0. In the first case, w.l.o.g. we may assume that $P_1 = \emptyset$ (otherwise work in Q_{k-1}). Let ρ_1' be the isomorphic copy of ρ_1 in Q_1. Let D be the cycle spanning all vertices of Q_0 and Q_1 obtained by joining ρ_1 and ρ_1'. In the second case, w.l.o.g. we may assume that $P_1 = \emptyset$. Let ρ_1' and ρ_2' be the isomorphic copies of ρ_1 and ρ_2, respectively, in Q_1. Let D_1 and D_2 be the cycles obtained by joining ρ_1 and ρ_1' and ρ_2 and ρ_2', respectively. Again, w.l.o.g. we may assume that $P_2 = \emptyset$. Choose edges f_1 and f_2 in Q_1 that lie in D_1 and D_2, respectively, and let f_1' and f_2' be the isomorphic copies of f_1 and f_2 in Q_2. By the induction hypothesis applied to $(\{f_1', f_2'\}, Q_2)$, there is a Hamiltonian cycle C_2 in Q_2 containing f_1' and f_2'. Join D_1, D_2 and C_2 using the bridges involving f_1 and f_1' and f_2 and $f2'$ to obtain the cycle D. Whatever the situation, we obtain the result using Lemma 2.

Suppose that $|P_0| = 2n - 1$. Let e and f be two edges of P_0. Applying the induction hypothesis to $(P_0 \setminus \{e, f\}, Q_0)$ yields a Hamlitonian cycle C_0 of Q_0 containing every edge of $P_0 \setminus \{e, f\}$. Suppose that C_0 contains at least one of e and f also. Now we proceed exactly as we did in the case above when $|P_0| = 2n-2$ and the edge (p, q) does not lie on (the previous cycle) C_0. Doing so, and then applying Lemma 2, yields the result. Hence, we may assume that both e and f do not appear in C_0. Consider e. Suppose that there is an edge of P_0 lying on C_0 and incident with e. Let e' and e'' be the edges of the maximal path ρ' of $\langle P_0 \rangle$ containing e that are incident with the terminal vertices of ρ'. Reapply the induction hypothesis to $(P_0 \setminus \{e', e''\}, Q_0)$ to obtain a Hamiltonian cycle C_0' of Q_0 containing every edge of $P_0 \setminus \{e', e''\}$. As above, we may assume that neither e' nor e'' lies on C_0'. Let ρ be the sub-path of C_0' joining the terminal vertices of ρ' and which contains no other vertex of ρ'. Let the terminal vertex of ρ' incident with e' (resp. e'') be c' (resp. c''), and let d' (resp. d'') be the other vertex incident with e' (resp. e''). W.l.o.g. we may assume that there is a sub-path of C_0' from c' to d' on which neither c'' nor d'' appears (in the alternative situation we proceed almost identically). There are 3 cases: $|\rho| = 1$; $|\rho| = 2$; and $|\rho| > 2$. These sub-cases can be visualised as in Fig. 4 (note that the sub-path $\rho' \setminus \{e', e''\}$ might consist of one vertex only). Note that none of the edges of C_0' incident with c' or c'' are in P_0 and that all edges of ρ', apart from e' and e'', lie on C_0'. Let a (resp. b) be the vertex of the sub-path of C_0' from c' to d' (resp. c'' to d'') and avoiding c'' (resp. c') that is adjacent to d'. In all cases, let ρ_1 be the path starting with the sub-path of C_0' from a to c' avoiding b, concatenated with the path ρ', and concatenated with the sub-path of C_0' from c'' to b avoiding a. If $|\rho| = 2$ then let x be the solitary internal vertex of ρ, and if $|\rho| > 2$ then let the vertex of ρ adjacent to c' (resp. c'') be x (resp. y). We now proceed essentially as we did in Case 2.c, Case 2.b and Case 1 of Lemma 6 to obtain the result.

So, we may assume that we have a Hamiltonian cycle C_0 of Q_0 containing all edges of $P_0 \setminus \{e, f\}$ so that the edges e and f do not lie on C_0 and are such that neither e nor f is incident with an edge of C_0 lying on P_0. We proceed as above for each situation as in Fig. 4. The result follows. □

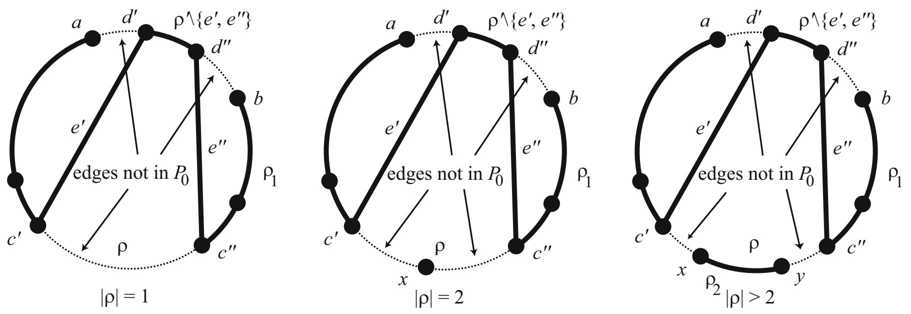


Fig. 4. The case when $|P_0| = 2n - 1$

4 Conclusions

A simple induction shows that we can select a set P of $4n - 2$ edges in Q_n^k, where $n \geq 2$ and $k \geq 3$, so that $\langle P \rangle$ consists of a set of vertex-disjoint paths and there exists a vertex x of Q_n^k so that all but 1 of x's neighbours in Q_n^k are incident with exactly 2 edges of P. Thus, the maximal size of a set P of edges of Q_n^k for which a version of Theorem 3 or Theorem 4 holds is at most $4n - 3$. It would be interesting to establish exactly where this threshold lies. We expect that given the more complex structure of the k-ary n-cube, the exact threshold will be much more difficult to obtain than it was for the hypercube.

As we mentioned earlier, there has been a significant amount of research undertaken as regards the necessity of the existence of Hamiltonian cycles in hypercubes and k-ary n-cubes either avoiding or containing prescribed sets of edges of a given size. The general question of given a set of edges of a hypercube or a k-ary n-cube (with no bound on the size of the set), does there exist a Hamiltonian cycle containing these edges, has yet to be considered as regards its computational complexity. It could well be that the proof of related complexity-theoretic results from [1,4] will provide an entry point into such an investigation.

References

1. Ashir, Y.A., Stewart, I.A.: Fault-tolerant embeddings of Hamiltonian circuits in k-ary n-cubes. SIAM Journal on Discrete Mathematics 15(3), 317–328 (2002)
2. Bose, B., Broeg, B., Kwon, Y., Ashir, Y.: Lee distance and topological properties of k-ary n-cubes. IEEE Transactions on Computers 44(8), 1021–1030 (1995)
3. Caha, R., Koubek, V.: Hamiltonian cycles and paths with a prescribed set of edges in hypercubes and dense sets. Journal of Graph Theory 51(2), 137–169 (2006)
4. Chan, M.Y., Lee, S.J.: On the existence of Hamiltonian circuits in faulty hypercubes. SIAM Journal on Discrete Mathematics 4(4), 511–527 (1991)
5. Dvořák, T.: Hamiltonian cycles with prescribed edges in hypercubes. SIAM Journal on Discrete Mathematics 19(1), 135–144 (2005)

6. Hsu, L.-H., Lin, C.-K.: Graph Theory and Interconnection Networks. CRC Press, Boca Raton (2009)
7. Gros, L.: Théorie du Baguenodier, Aimé Vingtrinier, Lyon (1872)
8. Stewart, I.A., Xiang, Y.: Bipanconnectivity and bipancyclicity in k-ary n-cubes. IEEE Transactions on Parallel and Distributed Systems 20(1), 25–33 (2009)
9. Wang, S., Li, J., Wang, R.: Hamiltonian cycles with prescribed edges in the 3-ary n-cube. Information Sciences 181(14), 3054–3065 (2011)

A Fast Parallel Algorithm for Finding a Most Reliable Source on a General Ring-Tree Graph with Unreliable Edges

Wei Ding[1] and Guoliang Xue[2]

[1] Zhejiang Water Conservancy and Hydropower College,
Hangzhou, Zhejiang 310000, China
dingweicumt@163.com

[2] Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-8809, USA
xue@asu.edu

Abstract. Given an unreliable communication network, we aim to find a most reliable source (MRS) on the network, which maximizes the expected number of nodes that are reachable from it. Although the problem of finding an MRS on general graphs is #P-hard, it is tractable in several types of sparse graphs. The ring-tree graph is such a kind of sparse graph that not only has the capability of failure tolerance but also holds an underlying tree topology which facilitates network administration. In this paper, we are concerned with unreliable general ring-tree graphs in which each edge has an independent operational probability while all nodes are immune to failures. We first design a complementary dynamic programming algorithm and then develop a parallel algorithm based on the underlying tree for finding an MRS on the network.

Keywords: Most reliable source, general ring-tree graph, complementary dynamic programming, parallel algorithm.

1 Introduction

A computer network or communication network is typically modeled as an undirected graph $G = (V, E)$, where V is a set of nodes that represent processing or switching elements and E is a set of edges that represent communication links [3]. Given any pair of nodes u and v, the communication between u and v is achieved by a u-to-v path. Network failures may occur to links or nodes [5,6,7,8,9,10,13,19]. As networks grow in size, they become increasingly vulnerable to failures. In the past decade, a large number of network reliability problems have been widely studied [1,2,11,14,15]. Many of them can be reduced to the computation of a most reliable source, defined in the following.

Given two nodes u and v in an unreliable communication network, we let $\Pr(u, v)$ denote the probability that a message can be transmitted correctly from u to v. The expected number of nodes reachable from u is called the *reachability* of u, denoted by $\mathrm{E}(u)$. Thus, we have

$$\mathrm{E}[u] = \sum_{v \in V} \Pr(u, v) \ . \tag{1}$$

A node that maximizes its reachability is called a *most reliable source* (MRS) of the network. The essence of computing an MRS of a given network is determining a node u^* in the network such that

$$\mathrm{E}[u^*] = \max_{u \in V} \sum_{v \in V} \Pr(u, v) \ . \tag{2}$$

As we all know, an MRS is a good candidate as the source for data broadcast in an unreliable network as the expected number of nodes reachable from an MRS is maximized. The problem of computing an MRS of an unreliable network is one of the network reliability problems, which has been studied widely in the past decade. Although this problem is #P-hard in general graphs [5,17], it is tractable in several types of sparse networks. Some papers are concerned with the case in which each link has an independent operational probability while all nodes are immune to failures. For tree networks, Melachrinoudis and Helander [13] presented a quadratic time algorithm and Xue [19] developed an improved linear time algorithm. For series-parallel graphs, Colbourn and Xue [6] devised a linear time algorithm. For ring graphs, Ding [7] gave a quadratic time algorithm. Also, Ding and Xue [10] studied another case in which each node has an independent probability of being faulty while all links are immune to failures and proposed a linear time algorithm for such tree networks.

The tree network is one of the most important network topologies as it has a sparse and recursive structure which facilitates administrating it. However, it has a principal weakness of low capability of failure tolerance under an unreliable setting, which stems from its low connectivity (there is a single path between its each node pair). A general network has a higher capability of failure tolerance, but it is hard to administrate it, e.g., it is #P-hard [5,17] to find its an MRS. The *ring-tree* network, formally defined in Sect. 2, is a good compromise between the network capability of failure tolerance and the ease of network administration. For instance, given a sample tree distinguished by bold edges in Fig. 1–(b), adding five dashed edges $\{a, e\}, \{f, g\}, \{h, i\}, \{k, l\}, \{m, n\}$ results in five embedded rings $abcdea, bfgb, chijc, jklj, dmnd$. In contrast, there is a single f-to-c path fbc in the original tree while there are four f-to-c paths $fbc, fgbc, fbaedc, fgbaedc$ in the resulting ring-tree. On the other hand, the ring-tree holds an underlying tree topology, which facilitates network administration. We can derive different ring-trees from any given tree by adding different edges to it. Furthermore, we refer readers to [8] for the detailed analysis on both the capability of failure tolerance of a general ring-tree graph and its reliability of communication.

In [9], Ding and Xue designed a fast Divide-and-Conquer algorithm for computing an MRS on *m–rings* graphs (a special type of ring-trees graph with an underlying topology of a strip graph, see Fig. 1–(a)) where each edge has an independent operational probability while all nodes are immune to failures.

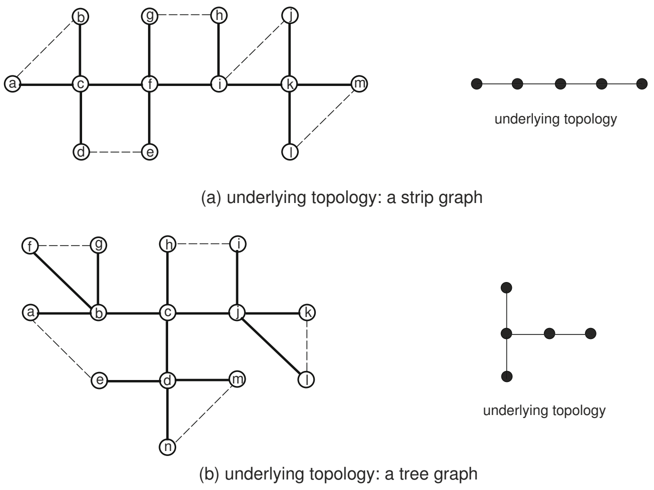

underlying topology
(a) underlying topology: a strip graph
underlying topology
(b) underlying topology: a tree graph

Fig. 1. Bold solid edges of two left-hand graphs form two sample tree networks. Adding several dashed edges to them yields two ring-tree networks, whose underlying topologies are obtained by shrinking every ring into a vertex and using an edge to represent the adjacency relationship of two rings, shown as right-hand graphs: (a) underlying topology is a strip graph; (b) underlying topology is a tree graph. It is evident that (a) is a special case of (b).

In this paper, we focus on a general ring-tree graph under a same unreliable setting as above, and propose an efficient parallel algorithm based on its underlying tree for finding its MRS with a time complexity of at most $O((\lambda^2 + \lambda\mu + 3\lambda + \mu)H(T))$.

The rest of this paper is organized as follows. In Sect. 2, we define the general ring-tree graph formally and some notations used frequently. In Sect. 3, we complete some fundamental preliminaries, including the decomposition scheme of ring-tree and several recurrence equations. In Sect. 4, we first design a complementary dynamic programming algorithm and then develop a parallel algorithm for finding an MRS of a given unreliable general ring-tree graph. In Sect. 5, we present some concluding remarks.

2 Definitions and Notations

Definition 1. *Let $T = (V(T), E(T))$ be an undirected tree. A general ring-tree graph $RT = (V(RT), E(RT))$ with an underlying topology T is constructed in the following way: (i) each node $v_i \in V(T)$ is expanded into an undirected ring $C_i = (V_i, E_i)$; (ii) each edge $e \in E(T)$ is removed. (see Fig. 1–(b))*

In this paper, we concern with an edge-weighted ring-tree as follows. Given any $i \in \{1, \ldots, |V(T)|\}$, every $e = \{x, y\} \in E_i$ is associated with a weight $p(e)$ representing the *edge operational probability* of e and two arcs of (x, y) from x to

y and (y, x) from y to x. The operational probabilities of (x, y) and (y, x) are both equal to the operational probability of $\{x, y\}$. As a consequence, we construct an edge-weighted ring $C_i = (V_i, E_i, p)$ and accordingly an edge-weighted ring-tree graph $RT = (V(RT), E(RT), p)$.

We can always take the underlying topology T as a rooted tree in this paper since an unrooted tree can be transformed into a rooted tree by designating any node of the tree as its *root*. For simplicity of presentation, we use numbers $i = 1, 2, \ldots, |V(T)|$ to label all nodes of T in the order of from bottom to root and from left to right on a level amongst T. Clearly $|V(T)|$ just represents the root of T. Let $f(i)$ denote the *parent* of i and $\kappa(i)$ denote the index of i in all *children* of $f(i)$ for every $i \in \{1, \ldots, V(T) - 1\}$. All leaves of T form a set L. Let $S(i)$ denote the set of all children of i for every $i \in \{1, \ldots, V(T)\} \setminus L$. By investigation, we discover that

$$|E(T)| = \sum_{i \in \{1, \ldots, |V(T)|\} \setminus L} |S(i)| \, . \tag{3}$$

In RT, given a node $i \in V(T)$ and its parent $f(i)$, we call $C_{f(i)}$ the *parent ring* of C_i and C_i a *child ring* of $C_{f(i)}$. A ring obtained by expanding a leaf of T is called a *leaf ring* of RT and the ring obtained by expanding the root $|V(T)|$ of T is called the *root ring* of RT. Given any ring C_i, the common node of C_i and $C_{f(i)}$ is both called *top joint* of C_i and *bottom joint* of $C_{f(i)}$. Note that the root ring has no top joint and every leaf ring has no bottom joint. Let $\alpha(i)$ denote the top joint of C_i and $\beta(i; k)$ denote the k–th bottom joint of C_i. The relationship of $\alpha(i) = \beta(f(i); \kappa(i))$ holds for every $i \in \{1, \ldots, |V(T)| - 1\}$.

Definition 2. *Given $i \in \{1, \ldots, |V(T)|\}$ and any pair of nodes $u, v \in V_i$, let $Q_i^+(u, v), Q_i^-(u, v)$ denote the probability that a message is correctly transmitted from u to v along C_i in the clockwise direction and in the counterclockwise direction respectively, and $Q_i(u, v)$ denote the probability that a message is correctly transmitted from u to v along C_i.*

For any node $i \in \{1, \ldots, |V(T)|\}$, the subtree of T rooted at i is denoted by T_i. The subgraph of RT constructed by T_i is denoted by $RT_i = (V(RT_i), E(RT_i))$. All nodes of RT outside RT_i form a set $\overline{V(RT_i)}$. In addition, we use $A \oplus B$ to denote the union of two disjoint sets A and B. Obviously, $V(RT_i) \oplus \overline{V(RT_i)} = V(RT)$ for any $i \in \{1, \ldots, |V(T)|\}$.

Definition 3. *Given a node $u \in V_i$, let $\mathcal{R}[u; C_i]$ denote the expected number of nodes in C_i other than u which are reached from u, $\mathcal{X}[u; C_i]$ denote the expected number of nodes in $V(RT_i)$ other than u which are reached from u, and $\mathcal{Y}[u; C_i]$ denote the expected number of nodes in $\overline{V(RT_i)}$ which are reached from u.*

For any $u \in V_i, i \in \{1, \ldots, |V(T)|\}$, we can formulate $\mathcal{R}[u; C_i]$ as Eq. (4), $\mathcal{X}[u; C_i]$ as Eq. (5) and $\mathcal{Y}[u; C_i]$ as Eq. (6) respectively according to their definitions in Definition 3. Note that $Q_i(u, v) = \Pr(u, v)$ for any $u, v \in V_i$.

$$\mathcal{R}[u; C_i] = \sum_{v \in V_i \setminus \{u\}} Q_i(u, v), \quad i \in \{1, \ldots, |V(T)|\} \, , \tag{4}$$

$$\mathcal{X}[u;C_i] = \sum_{v \in V(RT_i) \setminus \{u\}} \Pr(u,v), \quad i \in \{1,\ldots,|V(T)|\} , \tag{5}$$

$$\mathcal{Y}[u;C_i] = \sum_{v \in \overline{V(RT_i)}} \Pr(u,v), \quad i \in \{1,\ldots,|V(T)|\} . \tag{6}$$

3 Fundamental Preliminaries

It is easy to observe that $V(RT_i) = V_i$ when $i \in L$ and $\overline{V(RT_{|V(T)|})} = \emptyset$. In other cases, we can decompose $V(RT_i)$ and $\overline{V(RT_i)}$ recursively using the approach shown in Lemma 1. These decomposition schemes form the basis of our dynamic programming algorithm DMRS in Sect. 4.2.

Lemma 1. *For any $i \in \{1,\ldots,|V(T)|\} \setminus L$, we can decompose $V(RT_i)$ as*

$$V(RT_i) = V_i \oplus \left(\bigoplus_{k \in S(i)} (V(RT_k) \setminus \{\alpha(k)\}) \right) . \tag{7}$$

For any $i \in \{1,\ldots,|V(T)|-1\}$, we can decompose $\overline{V(RT_i)}$ as

$$\overline{V(RT_i)} = \left(V_{f(i)} \setminus \{\alpha(i)\}\right) \oplus \overline{V(RT_{f(i)})} \oplus \left(\bigoplus_{k \in S(f(i)) \setminus \{i\}} (V(RT_k) \setminus \{\alpha(k)\}) \right) . \tag{8}$$

Given a node $i \in \{1,\ldots,|V(T)|\}$, we can use the formula shown in Theorem 1 to compute $\mathrm{E}[u]$ for every $u \in V_i$. When $i \in L$, we conclude that $\mathcal{X}[u;C_i] = \mathcal{R}[u;C_i]$ from Eq. (4) and (5) together with the fact that $V(RT_i) = V_i$. When $i \notin L$, we can use the formula shown in Theorem 2 to compute $\mathcal{X}[u;C_i]$ for every $u \in V_i$. On the other hand, when $i = |V(T)|$, we conclude that $\mathcal{Y}[u;C_{|V(T)|}] = 0$ from Eq. (6) together with the fact that $\overline{V(RT_{|V(T)|})} = \emptyset$. When $i < |V(T)|$, we can use the formula shown in Theorem 3 to compute $\mathcal{Y}[u;C_i]$ for every $u \in V_i$.

Theorem 1. *For any $u \in V_i, i \in \{1,\ldots,|V(T)|\}$, we can compute* $\mathrm{E}[u]$ *by*

$$\mathrm{E}[u] = 1 + \mathcal{X}[u;C_i] + \mathcal{Y}[u;C_i] . \tag{9}$$

Proof. For any $u \in V_i, i \in \{1,\ldots,|V(T)|\}$, we have $V(RT) = V(RT_i) \oplus \overline{V(RT_i)} = \{u\} \oplus (V(RT_i) \setminus \{u\}) \oplus \overline{V(RT_i)}$. Combing Eq. (1), (5) and (6), we conclude that

$$\begin{aligned}
\mathrm{E}[u] &\overset{\text{Eq.(1)}}{=} \sum_{v \in V(RT)} \Pr(u,v) \\
&= \Pr(u,u) + \sum_{v \in V(RT_i) \setminus \{u\}} \Pr(u,v) + \sum_{v \in \overline{V(RT_i)}} \Pr(u,v) \\
&\overset{\text{Eq.(5),(6)}}{=} 1 + \mathcal{X}[u;C_i] + \mathcal{Y}[u;C_i] .
\end{aligned}$$

□

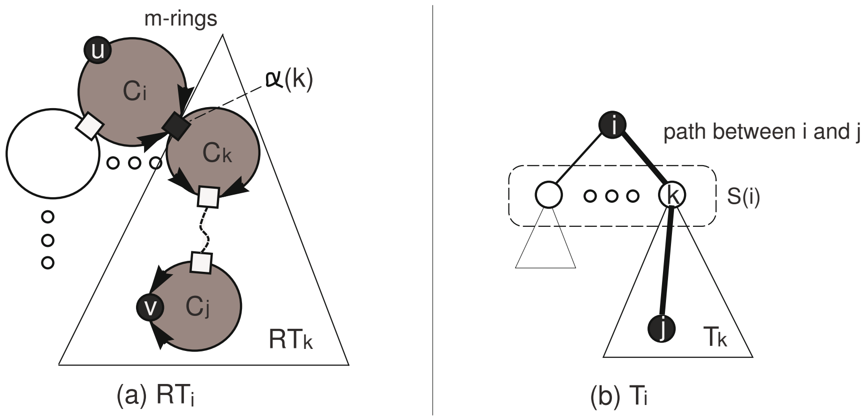


Fig. 2. Illustrate the proof of Theorem 2. In subfigure (b), the subtree T_i of T is shown. The unique path between two nodes i and j of T_i has m nodes, distinguished by bold edges. In subfigure (a), the corresponding sub-ring-tree RT_i of RT to T_i is shown. A symbol □ indicates a joint and specially symbol ■ indicates $\alpha(k)$, as well as the resulting m-rings on RT_i from the path is distinguished by color grey.

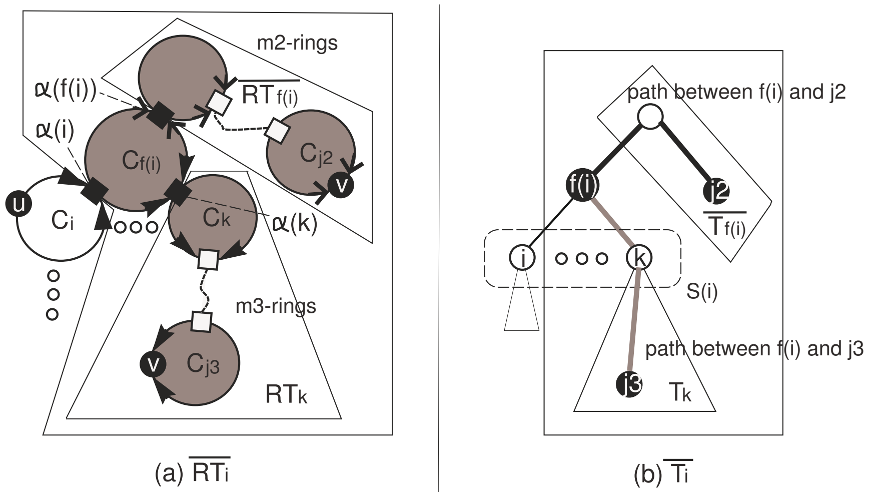


Fig. 3. Illustrate the proof of Theorem 3. In subfigure (b), the subgraph $\overline{T_i}$ of T is shown. The unique path between two nodes $f(i)$ and $j_2 \in \overline{V_{f(i)}}$ has m_2 nodes distinguished by bold black edges, and the unique path between two nodes $f(i)$ and $j_3 \in V_k, k \in S(f(i)) \setminus \{i\}$ has m_3 nodes distinguished by bold grey edges. In subfigure (a), the corresponding subgraph $\overline{RT_i}$ of RT to $\overline{T_i}$ is shown. A symbol □ indicates a joint and specially three symbols ■ indicate $\alpha(i), \alpha(f(i)), \alpha(k)$ respectively, as well as both the resulting m_2-rings on $\overline{RT_i}$ from the path with m_2 nodes and the resulting m_3-rings from the path with m_3 nodes are distinguished by color grey.

Theorem 2. *For any $u \in V_i, i \in \{1, \dots, |V(T)|\} \setminus L$, we compute $\mathcal{X}[u; C_i]$ by*

$$\mathcal{X}[u; C_i] = \mathcal{R}[u; C_i] + \sum_{k \in S(i)} Q_i(u, \alpha(k)) \cdot \mathcal{X}[\alpha(k); C_k] . \tag{10}$$

Proof. For every $i \in \{1, \dots, |V(T)|\} \setminus L$, we observe from Eq. (7) that any node $v \in V(RT_i)$ lies in V_i or one of $V(RT_k) \setminus \{\alpha(k)\}, k \in S(i)$. For any node $u \in V_i$, when $v \in V_i$, we have $\Pr(u, v) = Q_i(u, v)$. When $v \in V(RT_k) \setminus \{\alpha(k)\}, k \in S(i)$, without loss of generality, we suppose that $v \in V_j$ and the unique path between two indices i and j on T has m nodes. We can use the way in Definition 1 to construct an m–rings and apply the related method in [9] to this m–rings to obtain that $\Pr(u, v) = Q_i(u, \alpha(k)) \cdot \Pr(\alpha(k), v)$, see Fig. 2. Combing Eq. (4), (5) and (7), we conclude that

$$\begin{aligned}
\mathcal{X}[u; C_i] &\overset{\text{Eq.(5)}}{=} \sum_{v \in V(RT_i) \setminus \{u\}} \Pr(u, v) \\
&\overset{\text{Eq.(7)}}{=} \sum_{v \in V_i \setminus \{u\}} Q_i(u, v) + \sum_{k \in S(i)} \sum_{v \in V(RT_k) \setminus \{\alpha(k)\}} Q_i(u, \alpha(k)) \cdot \Pr(\alpha(k), v) \\
&\overset{\text{Eq.(4)}}{=} \mathcal{R}[u; C_i] + \sum_{k \in S(i)} \Big(Q_i(u, \alpha(k)) \cdot \sum_{v \in V(RT_k) \setminus \{\alpha(k)\}} \Pr(\alpha(k), v) \Big) \\
&= \mathcal{R}[u; C_i] + \sum_{k \in S(i)} Q_i(u, \alpha(k)) \cdot \mathcal{X}[\alpha(k); C_k] . \qquad \square
\end{aligned}$$

Theorem 3. *For any $u \in V_i, i \in \{1, \dots, |V(T)| - 1\}$, we compute $\mathcal{Y}[u; C_i]$ by*

$$\mathcal{Y}[u; C_i] = Q_i(u, \alpha(i)) \cdot \mathcal{Y}[\alpha(i); C_i] , \tag{11}$$

where

$$\begin{aligned}
\mathcal{Y}[\alpha(i); C_i] = \mathcal{R}[\alpha(i); C_{f(i)}] &+ Q_{f(i)}(\alpha(i), \alpha(f(i))) \cdot \mathcal{Y}[\alpha(f(i)); C_{f(i)}] \\
&+ \sum_{k \in S(f(i)) \setminus \{i\}} Q_{f(i)}(\alpha(i), \alpha(k)) \cdot \mathcal{X}[\alpha(k); C_k] .
\end{aligned} \tag{12}$$

Proof. For every $i \in \{|V(T)| - 1, \dots, 1\}$, we see from Eq. (8) that any node $v \in \overline{V(RT_i)}$ lies in $V_{f(i)} \setminus \{\alpha(i)\}$ or $\overline{V(RT_{f(i)})}$ or one of $V(RT_k) \setminus \{\alpha(k)\}, k \in S(f(i)) \setminus \{i\}$. For any node $u \in V_i$, u reach v via $\alpha(i)$. Without loss of generality, we suppose that $v \in V_{j_1}$ and the unique path between two indices i and j_1 on T has m_1 nodes. We can use the way in Definition 1 to construct an m_1–rings and apply the related method in [9] to this m_1–rings to obtain that $\Pr(u, v) = Q_i(u, \alpha(i)) \cdot \Pr(\alpha(i), v)$. Combing Eq. (6), we conclude that

$$\begin{aligned}
\mathcal{Y}[u; C_i] &\overset{\text{Eq.(6)}}{=} \sum_{v \in \overline{V(RT_i)}} \Pr(u, v) = \sum_{v \in \overline{V(RT_i)}} Q_i(u, \alpha(i)) \cdot \Pr(\alpha(i), v) \\
&= Q_i(u, \alpha(i)) \cdot \sum_{v \in \overline{V(RT_i)}} \Pr(\alpha(i), v) = Q_i(u, \alpha(i)) \cdot \mathcal{Y}[\alpha(i); C_i] .
\end{aligned}$$

When $v \in V_{f(i)} \setminus \{\alpha(i)\}$, we have $\Pr(\alpha(i), v) = Q_{f(i)}(\alpha(i), v)$. When $v \in \overline{V(RT_{f(i)})}$, we suppose that $v \in V_{j_2}$ and the unique path between two indices $f(i)$ and j_2 on T has m_2 nodes. We can use the way in Definition 1 to construct an m_2–rings and apply the related method in [9] to this m_2–rings to obtain that $\Pr(\alpha(i), v) = Q_{f(i)}(\alpha(i), \alpha(f(i))) \cdot \Pr(\alpha(f(i)), v)$, see Fig. 3. Likewise, when $v \in V(RT_k) \setminus \{\alpha(k)\}, k \in S(f(i)) \setminus \{i\}$, we suppose that $v \in V_{j_3}$, construct an m_3–rings, and obtain that $\Pr(\alpha(i), v) = Q_{f(i)}(\alpha(i), \alpha(k)) \cdot \Pr(\alpha(k), v)$, see Fig. 3. Combing Eq. (4) and (8), we conclude that

$$\begin{aligned}
\mathcal{Y}[\alpha(i); C_i] &= \sum_{v \in \overline{V(RT_i)}} \Pr(\alpha(i), v) \\
&\overset{\text{Eq.(8)}}{=} \sum_{v \in V_{f(i)} \setminus \{\alpha(i)\}} Q_{f(i)}(\alpha(i), v) + \sum_{v \in \overline{V(RT_{f(i)})}} Q_{f(i)}(\alpha(i), \alpha(f(i))) \cdot \Pr(\alpha(f(i)), v) \\
&\qquad + \sum_{k \in S(f(i)) \setminus \{i\}} \sum_{v \in V(RT_k) \setminus \{\alpha(k)\}} Q_{f(i)}(\alpha(i), \alpha(k)) \cdot \Pr(\alpha(k), v) \\
&\overset{\text{Eq.(4)}}{=} \mathcal{R}[\alpha(i); C_{f(i)}] + Q_{f(i)}(\alpha(i), \alpha(f(i))) \cdot \sum_{v \in \overline{V(RT_{f(i)})}} \Pr(\alpha(f(i)), v) \\
&\qquad + \sum_{k \in S(f(i)) \setminus \{i\}} \Big(Q_{f(i)}(\alpha(i), \alpha(k)) \cdot \sum_{v \in V(RT_k) \setminus \{\alpha(k)\}} \Pr(\alpha(k), v) \Big) \\
&= \mathcal{R}[\alpha(i); C_{f(i)}] + Q_{f(i)}(\alpha(i), \alpha(f(i))) \cdot \mathcal{Y}[\alpha(f(i)); C_{f(i)}] \\
&\qquad + \sum_{k \in S(f(i)) \setminus \{i\}} Q_{f(i)}(\alpha(i), \alpha(k)) \cdot \mathcal{X}[\alpha(k); C_k]\,. \qquad \square
\end{aligned}$$

Due to Theorem 2 and 3, we claim that we need get the related probability values on C_i before computing $\mathcal{X}[u; C_i]$ and $\mathcal{Y}[u; C_i]$ for every $i \in \{1, \ldots, |V(T)|\}$. Here we pick up the formulas in [7,9] to compute these values, see Lemma 2.

Lemma 2. *Given any ring $C_i = (V_i, E_i, p), i \in \{1, \ldots, |V(T)|\}$, we have*

$$Q_i(u, v) = Q_i^+(u, v) + Q_i^-(u, v) - Q_i^+(u, v) \cdot Q_i^-(u, v)\,. \tag{13}$$

4 The Parallel Algorithm

In this section, we will design a dynamic programming algorithm DMRS in Sect. 4.2 and then develop a parallel algorithm PMRS on basis of algorithm DMRS in Sect. 4.3 for finding an MRS on a general ring-tree graph with unreliable edges. First of all, we give a procedure RMRS for finding an MRS on a ring in Sect. 4.1, which will be invoked by algorithm DMRS and PMRS.

4.1 An MRS on a Ring

Given any ring $C_i = (V_i, E_i, p), i \in \{1, \ldots, |V(T)|\}$, for any $u, v \in V_i$, Lemma 2 implies that we need compute $Q_i^+(u, v)$ and $Q_i^-(u, v)$ so as to compute $Q_i(u, v)$.

Combing Eq. (4), this directly leads us to the following procedure RMRS, whose time complexity is $O(|V_i|^2)$, see [7,9] for more related details.

Procedure RMRS:

Input: An edge-weighted ring $C_i = (V_i, E_i, p)$.
Output: All of $\mathcal{R}[u; C_i], u \in V_i$, all of $Q_i(\beta(i;k), \alpha(i)), k \in \{1, \ldots, |S(i)|\}$ and all of $Q_i(\beta(i;k_1), \beta(i;k_2)), k_1 \neq k_2 \in \{1, \ldots, |S(i)|\}$;

```
for every $u \in V_i$ do
  Compute all of $Q_i^+(u,v), v \in V_i$ in the clockwise direction;
  Compute all of $Q_i^-(u,v), v \in V_i$ in the counterclockwise direction;
  Compute all of $Q_i(u,v), v \in V_i$ using Eq. (13) and $\mathcal{R}[u; C_i]$ using Eq. (4);
end for
```

4.2 Dynamic Programming Algorithm

In this subsection, we will design a dynamic programming algorithm based on T using the technique of complementary dynamic computing in [10] for finding an MRS of a given general ring-tree graph RT with unreliable edges.

The essence of Theorem 1 provides us with a way of finding an MRS of RT. For every $i \in \{1, \ldots, |V(T)|\}$, we can first compute $\mathcal{X}[u; C_i], \mathcal{Y}[u; C_i]$ and then $\mathrm{E}[u]$ for all $u \in V_i$, finally determine the maximum of all $\mathrm{E}[u]$. This maximum corresponds to an MRS of RT. Hence, the key task is to compute $\mathcal{X}[u; C_i]$ and $\mathcal{Y}[u; C_i]$ for all $u \in V_i$ for every $i \in \{1, \ldots, |V(T)|\}$. Theorem 2 implies that we can compute all of $\mathcal{X}[u; C_i], u \in V_i$ using Eq. (10) for every $i \in \{1, \ldots, |V(T)|\} \setminus L$ by a bottom-up dynamic programming based on T. Theorem 3 implies that we can first compute $\mathcal{Y}[\alpha(i); C_i]$ using Eq. (12) and then all of $\mathcal{Y}[u; C_i], u \in V_i$ using Eq. (11) for every $i \in \{|V(T)| - 1, \ldots, 1\}$ by a top-down dynamic programming based on T. This forms our *complementary dynamic programming algorithm* DMRS, whose time complexity is shown in Theorem 4.

Algorithm DMRS:

Input: An edge-weighted ring-tree $RT = (V(RT), E(RT), p)$ with an underlying topology $T = (V(T), E(T))$.
Output: All of $\mathrm{E}[u], u \in V(RT)$.

```
Step_1 {$Initialize$}
    Use procedure RMRS to compute all of $\mathcal{R}[u; C_i], u \in V_i$ and
    related values for every $i \in \{1, 2, \ldots, |V(T)|\}$;
    for $i$ from 1 up to $|V(T)|$ do
      Set $\mathcal{X}[u; C_i] \leftarrow \mathcal{R}[u; C_i], \mathcal{Y}[u; C_i] \leftarrow 0$ for all $u \in V_i$;
    endfor
Step_2 {$Bottom-up\ dynamic\ programming\ on\ T$}
    for $i$ from 1 up to $|V(T)|$ do
      if $i \in L$ then break;
      else Compute $\mathcal{X}[u; C_i]$ by Eq. (10) for all $u \in V_i$; endif
```

```
        endfor
Step_3 {Top-down dynamic programming on $T$}
        for $i$ from $|V(T)|$ down to 1 do
          if $i = |V(T)|$ then break;
          else First compute $\mathcal{Y}[\alpha(i); C_i]$ by Eq. (12) and then
                $\mathcal{Y}[u; C_i]$ by Eq. (11) for all $u \in V_i$;
          endif
        endfor
Step_4 {Compute the reachability of each node}
        for $i$ from 1 up to $|V(T)|$ do
          Compute E$[u]$ by Eq. (9) for all $u \in V_i$;
        endfor
```

Theorem 4. *Given an edge-weighted ring-tree graph $RT = (V(RT), E(RT), p)$ with an underlying topology $T = (V(T), E(T))$, algorithm* DMRS *can find an MRS on RT correctly, with a time complexity of $O((\lambda^2 + 4\lambda + \mu)|V(T)|)$ where $\lambda = \max_{i \in \{1,\ldots,|V(T)|\}} |V_i|$ and $\mu = \max_{i \in \{1,\ldots,|V(T)|\} \setminus L} |S(i)|$.*

Proof. Step_1 of algorithm DMRS, for every $i = 1, \ldots, |V(T)|$, first takes $O(|V_i|^2)$ time to use procedure RMRS and then $O(|V_i|)$ time to initialize all of $\mathcal{X}[u; C_i]$ and $\mathcal{Y}[u; C_i]$. Thus, the running time of Step_1 is

$$\sum_{i=1}^{|V(T)|} O(|V_i|^2) + \sum_{i=1}^{|V(T)|} O(|V_i|) \leq O((\lambda^2 + \lambda)|V(T)|) .$$

Step_2 of algorithm DMRS, for every $i = 1, \ldots, |V(T)|$, spends $O(1)$ time to break when $i \in L$ and $O(|S(i)|)$ time to compute $\mathcal{X}[u; C_i]$ using Eq. (10) for every $u \in V_i$ when $i \notin L$. Thus, the running time of Step_2 is

$$\begin{aligned}
& \sum_{i \in L} O(1) + \sum_{i \in \{1,\ldots,|V(T)|\} \setminus L} \sum_{u \in V_i} O(|S(i)|) \\
= \; & O(|L|) + \sum_{i \in \{1,\ldots,|V(T)|\} \setminus L} O(|S(i)| \cdot |V_i|) \\
\leq \; & O(|V(T)|) + \sum_{i \in \{1,\ldots,|V(T)|\} \setminus L} O(\lambda |S(i)|) \\
\overset{\text{Eq.(3)}}{=} \; & O(|V(T)|) + O(\lambda |E(T)|) = O(\lambda |V(T)|) .
\end{aligned}$$

Step_3 of algorithm DMRS, for every $i = |V(T)|, \ldots, 1$, spends $O(1)$ time to break when $i = |V(T)|$, as well as $O(|S(f(i))|)$ time to compute $\mathcal{Y}[\alpha(i); C_i]$ using Eq. (12) and then $O(1)$ time to compute $\mathcal{Y}[u; C_i]$ using Eq. (11) for every $u \in V_i$ when $i < |V(T)|$. Combing with the fact that $\sum_{i=1}^{|V(T)|-1} |S(f(i))| = \sum_{i \in \{1,\ldots,|V(T)|\} \setminus L} |S(i)|^2$, we conclude that the running time of Step_3 is

$$\begin{aligned} & O(1)+\sum_{i=1}^{|V(T)|-1}\Big(O(|S(f(i))|)+\sum_{u\in V_i}O(1)\Big) \\ = & \sum_{i\in\{1,\ldots,|V(T)|\}\setminus L} O(|S(i)|^2)+\sum_{i=1}^{|V(T)|-1} O(|V_i|) \\ \le & \sum_{i\in\{1,\ldots,|V(T)|\}\setminus L} O(\mu|S(i)|)+O(\lambda|V(T)|) \\ \overset{\text{Eq.(3)}}{=} & O(\mu|E(T)|)+O(\lambda|V(T)|)=O((\mu+\lambda)|V(T)|)\,. \end{aligned}$$

Step_4 of algorithm DMRS, for every $i = 1, \ldots, |V(T)|$, spends $O(1)$ time to compute E$[u]$ using Eq. (9) for every $u \in V_i$. Thus, the running time of Step_4 is $\sum_{i=1}^{|V(T)|}\sum_{u\in V_i} O(1) = \sum_{i=1}^{|V(T)|} O(|V_i|) \le O(\lambda|V(T)|)$.

Therefore, it follows that the total time complexity of algorithm DMRS is $O((\lambda^2 + 4\lambda + \mu)|V(T)|)$. □

4.3 Parallel Algorithm

The design and analysis of parallel algorithm has been extensively studied in past decades [4,12,16,18,20]. In this subsection, we will develop a parallel algorithm on basis of our dynamic programming algorithm DMRS for finding an MRS of a given general ring-tree graph RT with unreliable edges.

Let $H(T)$ denote the height of T and h denote the variable of current height. Here we label the bottom level of T as the 1–th level. Let $V(h)$ denote the set of nodes on the h–level of T and $K = \max_{h=1,\ldots,H(T)} |V(h)|$. Let $\mathbb{M}_1, \mathbb{M}_2, \ldots, \mathbb{M}_K$ represent a group of K identical processors. For every $h \in \{1, \ldots, H(T)\}$, since $|V(h)| \le K$, we can assign one of $\mathbb{M}_1, \mathbb{M}_2, \ldots, \mathbb{M}_K$ individually to accomplish related work on C_i for each $i \in V(h)$. Consequently, some or all of $\mathbb{M}_1, \mathbb{M}_2, \ldots, \mathbb{M}_K$ can be assigned to execute parallel computing on every level of T.

As discussed in Sect. 2, all nodes of T have been labeled by $1, 2, \ldots, |V(T)|$ from bottom to root and from left to right on a level amongst T. All node labels on the h–level of T with $h = 2, \ldots, H(T)$ minus $\sum_{k=1}^{h-1} |V(k)|$ are $1, \ldots, |V(h)|$. Let $\pi(i)$ be the index of processor assigned to $C_i, i \in \{1, \ldots, |V(T)|\}$ so that $\pi(i) = i$ when $i \in V(1)$ and $\pi(i) = i - \sum_{k=1}^{h-1} |V(k)|$ when $i \in V(h), h \in \{2, \ldots, H(T)\}$. Thus, a sequence of $\pi(i), i \in \{1, \ldots, |V(T)|\}$ are listed as follows.

$$\begin{aligned} h=1: \;& \pi(1)=1, \pi(2)=2, \ldots, \pi(|V(1)|)=|V(1)|; \\ h=2: \;& \pi(|V(1)|+1)=1, \\ & \pi(|V(1)|+2)=2, \\ & \ldots\ldots \\ & \pi(|V(1)|+|V(2)|)=|V(2)|; \\ \vdots\; & \;\vdots \end{aligned}$$

$$\begin{aligned}
h = H(T)-1: \quad & \pi(\textstyle\sum_{k=1}^{H(T)-2} |V(k)|+1) = 1,\\
& \pi(\textstyle\sum_{k=1}^{H(T)-2} |V(k)|+2) = 2,\\
& \ldots\ldots\\
& \pi(\textstyle\sum_{k=1}^{H(T)-1} |V(k)|) = |V(H(T)-1)|;\\
h = H(T): \quad & \pi(\textstyle\sum_{k=1}^{H(T)-1} |V(k)|+1) = \pi(|V(T)|) = 1.
\end{aligned}$$

Based on discussions above, we replace the one-by-one computing of a single processor on every level of T in algorithm DMRS with the parallel computing of multiple processors. The details are presented as follows. First, on every level $h = 1, \ldots, H(T)$, each of $\mathbb{M}_{\pi(i)}, i \in V(h)$ uses procedure RMRS to compute all of $\mathcal{R}[u; C_i], u \in V_i$ and related probability values simultaneously as well as initializes $\mathcal{X}[u; C_i] = \mathcal{R}[u; C_i]$ and $\mathcal{Y}[u; C_i] = 0$. Next, on every level $h = 1, \ldots, H(T)$, each of $\mathbb{M}_{\pi(i)}, i \in L$ does not work and each of $\mathbb{M}_{\pi(i)}, i \notin L$ computes all of $\mathcal{X}[u; C_i], u \in V_i$ using Eq. (10). Next, $\mathbb{M}_{\pi(|V(T)|)}$ does not work when $h = H(T)$. On every level $h = H(T)-1, \ldots, 1$, all of $\mathbb{M}_{\pi(i)}, i \in V(h)$ compute $\mathcal{Y}[\alpha(i); C_i]$ using Eq. (12) and then all of $\mathcal{Y}[u; C_i], u \in V_i$ using Eq. (11). Finally, on every level $h = 1, \ldots, H(T)$, each of $\mathbb{M}_{\pi(i)}, i \in V(h)$ computes all of $\mathrm{E}[u], u \in V_i$ using Eq. (9). This forms our parallel algorithm PMRS, whose time complexity is shown in Theorem 5.

```
Algorithm PMRS
Input: An edge-weighted ring-tree $RT = (V(RT), E(RT), p)$ with an
underlying topology $T = (V(T), E(T))$.
Output: All of $\mathrm{E}[u], u \in V(RT)$.
Step_1 {Initialize}
        for $h$ from 1 up to $H(T)$ do
           Each of $\mathbb{M}_{\pi(i)}, i \in V(h)$ uses procedure RMRS to compute all of
           $\mathcal{R}[u; C_i], u \in V_i$ and related values, and sets $\mathcal{X}[u; C_i] \leftarrow \mathcal{R}[u; C_i]$,
           $\mathcal{Y}[u; C_i] \leftarrow 0$ for all $u \in V_i$;
        endfor
Step_2 {From bottom to top on $T$}
        for $h$ from 1 up to $H(T)$ do
           Each of $\mathbb{M}_{\pi(i)}, i \in V(h) \setminus L$ computes $\mathcal{X}[u; C_i]$ by Eq. (10) for
           all $u \in V_i$, while each of $\mathbb{M}_{\pi(i)}, i \in V(h) \cap L$ does not work;
        endfor
Step_3 {From top to bottom on $T$}
        for $h$ from $H(T)$ down to 1 do
           When $h = H(T)$, $\mathbb{M}_{\pi(|V(T)|)}$ does not work; when $h \neq H(T)$,
           each of $\mathbb{M}_{\pi(i)}, i \in V(h)$ first computes $\mathcal{Y}[\alpha(i); C_i]$ by Eq. (12)
           and then $\mathcal{Y}[u; C_i]$ by Eq. (11) for all $u \in V_i$;
        endfor
Step_4 {Compute the reachability of each node}
        for $h$ from 1 up to $H(T)$ do
           Each of $\mathbb{M}_{\pi(i)}, i \in V(h)$ computes $\mathrm{E}[u]$ by Eq. (9) for all $u \in V_i$;
        endfor
```

Theorem 5. *Given an edge-weighted ring-tree $RT = (V(RT), E(RT), p)$ with an underlying topology $T = (V(T), E(T))$, algorithm* PMRS *can find an MRS on RT correctly, with a time complexity of $O((\lambda^2 + \lambda\mu + 3\lambda + \mu)H(T))$ where $\lambda = \max_{i\in\{1,\dots,|V(T)|\}} |V_i|$ and $\mu = \max_{i\in\{1,\dots,|V(T)|\}\setminus L} |S(i)|$.*

Proof. In Step_1 of algorithm PMRS, on every $h = 1, \dots, H(T)$, each of $\mathbb{M}_{\pi(i)}, i \in V(h)$ first spends $O(|V_i|^2)$ time to use procedure RMRS once and then $O(|V_i|)$ time to accomplish all initializations. The time occupied on the h–level of T is decided by the maximum of all $O(|V_i|^2) + O(|V_i|), i \in V(h)$. Thus, the running time of Step_1 is

$$\sum_{h=1}^{H(T)} \max_{i\in V(h)} (O(|V_i|^2) + O(|V_i|)) \le \sum_{h=1}^{H(T)} O(\lambda^2 + \lambda) = O((\lambda^2 + \lambda)H(T)) .$$

In Step_2 of algorithm PMRS, on every $h = 1, \dots, H(T)$, all of $\mathbb{M}_{\pi(i)}, i \in V(h) \cap L$ do not work and each of $\mathbb{M}_{\pi(i)}, i \in V(h) \setminus L$ spends $O(|S(i)|)$ time to compute $\mathcal{X}[u; C_i]$ using Eq. (10) for every $u \in V_i$. The time occupied on the h–level of T is decided by the maximum of all $\sum_{u\in V_i} O(|S(i)|), i \in V(h) \setminus L$. Thus, the running time of Step_2 is

$$\sum_{h=1}^{H(T)} \Big(\max_{i\in V(h)\setminus L} \sum_{u\in V_i} O(|S(i)|) \Big) = \sum_{h=1}^{H(T)} \max_{i\in V(h)\setminus L} O(|S(i)| \cdot |V_i|) \le O(\lambda\mu H(T)) .$$

In Step_3 of algorithm PMRS, $\mathbb{M}_{\pi(|V(T)|)}$ does not work. On every $h = H(T) - 1, \dots, 1$, each of $\mathbb{M}_{\pi(i)}, i \in V(h)$ spends $O(|S(f(i))|)$ time to compute $\mathcal{Y}[\alpha(i); C_i]$ using Eq. (12) and then $O(1)$ time to compute $\mathcal{Y}[u; C_i]$ using Eq. (11) for every $u \in V_i$. The time occupied on the h–level of T is decided by the maximum of all $O(|S(f(i))|) + \sum_{u\in V_i} O(1), i \in V(h)$. Combing with the fact that $\max_{i\in V(k)} |S(f(i))| = \max_{i\in V(k+1)\setminus L} |S(i)|$ for every $k = 1, \dots, H(T) - 1$, we conclude that the running time of Step_3 is

$$\begin{aligned}
&\sum_{h=1}^{H(T)-1} \max_{i\in V(h)} \Big(O(|S(f(i))|) + \sum_{u\in V_i} O(1) \Big) \\
&\le \sum_{h=1}^{H(T)-1} \max_{i\in V(h)} O(|S(f(i))|) + \sum_{h=1}^{H(T)-1} \Big(\max_{i\in V(h)} \sum_{u\in V_i} O(1) \Big) \\
&= \sum_{h=2}^{H(T)} \max_{i\in V(h)\setminus L} O(|S(i)|) + \sum_{h=1}^{H(T)-1} \max_{i\in V(h)} O(|V_i|) \\
&\le O(\mu H(T)) + O(\lambda H(T)) = O((\mu + \lambda)H(T)) .
\end{aligned}$$

In Step_4 of algorithm PMRS, on every $h = 1, \dots, H(T)$, each of $\mathbb{M}_{\pi(i)}, i \in V(h)$ spends $O(1)$ time to compute $\mathrm{E}[u]$ using Eq. (9) for every $u \in V_i$. The time occupied on the h–level of T is decided by the maximum of all $O(|V_i|), i \in V(h)$. Thus, the running time of Step_4 is $\sum_{h=1}^{H(T)} \max_{i\in V(h)} O(|V_i|) \le O(\lambda H(T))$.

Therefore, it follows that the total time complexity of algorithm PMRS is $O((\lambda^2 + \lambda\mu + 3\lambda + \mu)H(T))$. □

5 Concluding Remarks

In this paper, given an unreliable general ring-tree network RT where each edge has an independent operational probability and all nodes are immune to failures and its underlying tree topology T, we have designed a dynamic programming algorithm DMRS and then a parallel algorithm PMRS based on T for finding an MRS on RT. According to Theorem 4 and 5, we can infer that PMRS is better than DMRS in general with respect to the time complexity of algorithm.

Likewise, for an m–rings graph, we can develop a parallel algorithm BMRS based on the *binary division tree* (BDT) from the Divide-and-Conquer algorithm in [9]. This BDT has $\lceil \log m \rceil$ levels. On the other hand, this m–rings graph can be transformed into a special ring-tree graph by designating the $\lceil \frac{m}{2} \rceil$–th node of its underlying strip topology as the root and transforming the strip into a $\wedge$–type tree. The $\wedge$–type tree has $\lceil \frac{m}{2} \rceil$ levels and its every level other than the top level has exact two nodes. Hence, we can use algorithm PMRS based on the $\wedge$–type tree to find an MRS on an unreliable m–rings graph. It always holds that $\lceil \log m \rceil \leq \lceil \frac{m}{2} \rceil$ when $m \geq 4$. As long as adequate processors are provided to execute parallel computing, PMRS is no better than than BMRS with respect to the time complexity of algorithm when $m \geq 4$.

All of discussions above on parallel algorithms deal with the case of adequate processors provided for parallel computing. However, provided that available processors are inadequate, e.g., $K < \max_{h \in \{1,\ldots,H(T)\}} |V_h|$, it is also an interesting topic how to devise an efficient parallel algorithm with a low time complexity executed by these K processors.

References

1. Ball, M.O., Lin, F.L.: A Reliability Model Applied to Emergency Service Vehicle Location. Oper. Res. 41(1), 18–36 (1993)
2. Ball, M.O., Provan, J.S., Shier, D.R.: Reliability Covering Problems. Networks 21(3), 345–357 (1991)
3. Bondy, J.A., Murty, U.S.R.: Graph Theory with Application. Macmillan, London (1976)
4. Censor, Y., Gordon, D., Gordon, R.: Component Averaging: an Efficient Iterative Parallel Algorithm for Large and Sparse Unstructured Problems. Parallel Computing 27(6), 777–808 (2001)
5. Colbourn, C.J.: The Combinatorics of Network Reliability. Oxford University Press, New York (1987)
6. Colbourn, C.J., Xue, G.: A Linear Time Algorithms for Computing the Most Reliable Source on a Series-Parallel Graph with Unreliable Edges. Theor. Comput. Sci. 209, 331–345 (1998)
7. Ding, W.: Computing the Most Reliable Source on Stochastic Ring Networks. In: 2009 WRI World Congress on Software Engineering, Xiamen, China, May 19–21, vol. 1, pp. 345–347 (2009)
8. Ding, W.: Embedded-Rings-Based Survivable Networks. In: 2010 International Conference on Industrial and Information Systems, Dalian, China, July 10–11, vol. 2, pp. 412–415 (2010)

9. Ding, W., Xue, G.: A divide-and-conquer algorithm for computing a most reliable source on an unreliable ring-embedded tree. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 268–280. Springer, Heidelberg (2010)
10. Ding, W., Xue, G.: A Linear Time Algorithm for Computing a Most Reliable Source on a Tree Network with Faulty Nodes. Theor. Comput. Sci. 412, 225–232 (2011)
11. Eiselt, H.A., Gendreau, M., Laporte, G.: Location of Facilities on a Network Subject to a Single-Edge Failure. Networks 22(3), 231–246 (1992)
12. Luby, M.: A Simple Parallel Algorithm for the Maximal Independent Set Problem. In: Proceedings of the seventeenth annual ACM symposium on Theory of computing (STOC 1985), Toronto, Canada, pp. 1–10 (1985)
13. Melachrinoudis, E., Helander, M.E.: A Single Facility Location Problem on a Tree with Unreliable Edges. Networks 27(3), 219–237 (1996)
14. Mirchandani, P.B., Odoni, A.R.: Locations of Medians on Stochastic Networks. Transport. Sci. 13, 85–97 (1979)
15. Nel, L.D., Colbourn, C.J.: Locating a Broadcast Facility in an Unreliable Network. INFOR 28, 363–379 (1990)
16. Pardalos, P.M., Xue, G., Panagiotopoulos, P.D.: Parallel Algorithms for Global Optimization Problems. In: Ferreira, A., Pardalos, P.M. (eds.) SCOOP 1995. LNCS, vol. 1054, pp. 232–247. Springer, Heidelberg (1996)
17. Shier, D.R.: Network Reliability and Algebraic Structure. Oxford University Press, New York (1991)
18. Sun, X.H., Rover, D.T.: Scalability of Parallel Algorithm-Machine Combinations. IEEE Transactions on Parallel and Distributed Systtem 5(6), 599–613 (1994)
19. Xue, G.: Linear Time Algorithms for Computing the Most Reliable Source on an Unreliable Tree Network. Networks 30(1), 37–45 (1997)
20. Zhang, T.Y., Suen, C.Y.: A Fast Parallel Algorithm for Thinning Digital Patterns. Communications of the ACM 27(3), 236–239 (1984)

Restricted Edge Connectivity of Harary Graphs[⋆]

Qinghai Liu, Xiaohui Huang, and Zhao Zhang[⋆⋆]

College of Mathematics and System Sciences, Xinjiang University,
Urumqi, Xinjiang, 830046, People's Republic of China

Abstract. An edge subset F of a connected graph $G = (V, E)$ is a k-restricted edge cut if $G - F$ is disconnected, and every component of $G - F$ has at least k vertices. The k-restricted edge connectivity of G, denoted by $\lambda_k(G)$, is the cardinality of a minimum k-restricted edge cut. By the current studies on λ_k, it can be seen that the larger λ_k is, the more reliable the graph is. Hence one expects λ_k to be as large as possible. A possible upper bound for λ_k is ξ_k defined as $\xi_k(G) = \min\{\omega(S) : \emptyset \neq S \subset V(G), |S| = k$ and $G[S]$ is connected$\}$, where $\omega(S)$ is the number of edges with one end in S and the other end in $V(G) \setminus S$, and $G[S]$ is the subgraph of G induced by S. A graph G is called λ_k-optimal if $\lambda_k(G) = \xi_k(G)$. A natural question is whether there exists a graph G which is λ_k-optimal for any $k \leq |V(G)|/2$. In this paper, we show that except for two cases, the Harary graph has this property.

1 Introduction

A network can be modelled as a graph $G = (V, E)$. A classic measure of network reliability is the edge connectivity $\lambda(G)$. In general, the larger $\lambda(G)$ is, the more reliable the network is [2]. It is well known that $\lambda(G) \leq \delta(G)$, where $\delta(G)$ is the minimum degree of G. Hence a graph G is called *maximally edge connected* or λ-*optimal* if $\lambda(G) = \delta(G)$.

For further study, the concept of restricted edge connectivity was proposed by Esfahanian and Hakimi [6], and then generalized to k-restricted edge connectivity by Fàbrega and Fiol [7]. An edge set $F \subset E$ is a *k-restricted edge cut* of a connected graph G if $G - F$ is disconnected and every component of $G - F$ has at least k vertices. The *k-restricted edge connectivity* of G, denoted by $\lambda_k(G)$, is the cardinality of a minimum k-restricted edge cut. Clearly, $\lambda_1(G) = \lambda(G)$. For simplicity, a *minimum* k-restricted edge cut is abbreviated as a λ_k-*cut*. Not all connected graphs have λ_k-cuts [4,6,13,19]. Those graphs which do have λ_k-cuts are said to be λ_k-*connected*.

In view of current studies in this aspect [5,11,15,17], it seems that the larger $\lambda_k(G)$ is, the more reliable the network is. So, we expect $\lambda_k(G)$ to be as large as possible. The optimization of $\lambda_k(G)$ requires an upper bound first.

⋆ This research is supported by NSFC (10971255, 61063005), Program for New Century Excellent Talents in University (NCET-08-0921), and The Project-sponsored by SRF for ROCS, SEM.

⋆⋆ Corresponding author.

For $S_1, S_2 \subset V(G)$, denote by $[S_1, S_2]_G$ the set of edges with one end in S_1 and the other end in S_2. For a vertex set $S \subseteq V(G)$, $G[S]$ is the subgraph of G induced by S, $\overline{S} = V(G)\backslash S$ is the complement of S. Denote by $\omega_G(S) = |[S, \overline{S}]_G|$ the number of edges between S and $\overline{S}$. When the graph under consideration is obvious, we omit the subscript G. Let

$$\xi_k(G) = \min\{\omega(S) : \emptyset \neq S \subset V(G), |S| = k \text{ and } G[S] \text{ is connected}\}.$$

Clearly, $\xi_1(G) = \delta(G)$, and thus the upper bound for $\lambda(G)$ is exactly $\lambda_1(G) \leq \xi_1(G)$. In [6], Esfahanian and Hakimi proved that $\lambda_2(G) \leq \xi_2(G)$ as long as G is λ_2-connected. In [4], Bonsma et al. proved that $\lambda_3(G) \leq \xi_3(G)$ as long as G is λ_3-connected. For $k \geq 4$, the inequality $\lambda_k(G) \leq \xi_k(G)$ is no longer true in general [19]. In [19], Zhang and Yuan showed that $\lambda_k(G) \leq \xi_k(G)$ for any $k \leq \delta(G) + 1$. A graph G is called λ_k*-optimal* if $\lambda_k(G) = \xi_k(G)$. For the studies on λ_k-optimal graphs, we refer the reader to the nice survey [10] and references therein.

We are interested in finding a graph which is λ_k-optimal for all $k \leq |V(G)|/2$. In this paper, we show that except for two cases, the Harary graph has this property. The reason why we consider Harary graph is because it is the first graph which was proved to have the highest possible connectivity over all graphs with given order and size [8]. Later studies show that they are also optimal with respect to some other measures of reliability [5,16].

A Harary graph $H_{m,d}$ has vertex set $\{0, 1, ..., m-1\}$. According to the parities of m and d, there are three types of Harary graphs. In the following, additions are all taken modulo m.

Type 1. When d is even, suppose $d = 2r$. Two vertices i and j of $H_{m,2r}$ are adjacent if and only if $|i - j| \leq r$.

Type 2. When d is odd and m is even, suppose $d = 2r + 1$. Then $H_{m,d}$ is obtained from $H_{m,2r}$ by adding edges $\{(i, i + \frac{m}{2}) : i = 0, 1, \ldots, \frac{m}{2} - 1\}$.

Type 3. When d and m are both odd, suppose $d = 2r+1$. Then $H_{m,d}$ is obtained from $H_{m,2r}$ by adding edges $\{(i, i + (m + 1)/2) : i = 0, 1, \ldots, (m - 3)/2\} \cup \{(0, (m - 1)/2)\}$.

Those edges in $H_{m,d}$ which are not in $H_{m,2r}$ are called *diagonals.* The classic result on Harary graphs is that they are maximally vertex connected (that is, $\kappa(G) = \delta(G)$, where $\kappa(G)$ is the vertex connectivity of G), and hence maximally edge connected. As consequences of [9,12], in which circulant graphs are studied, the first and the second types of Harary graphs are λ_2-optimal and λ_3-optimal. The λ_2-optimality of Harary graphs (including the third type) can also be found in [5]. In this paper, we determine $\lambda_k(G)$ and $\xi_k(G)$ for every Harary graph $G = H_{m,d}$ and every positive integer $k \leq m/2$. As a consequence, every Harary graph is λ_k-optimal except for two cases (Theorem 1 and Theorem 2).

Terminologies and notation not defined here are referred to [3].

2 Preliminaries

In this section, the addition is always assumed to be modulo $|V(G)|$. In our proofs, we determine the minimum value of $\omega(S)$ for vertex set S with $|S| = k \leq \lfloor \frac{|V(G)|}{2} \rfloor$. Then the values of $\lambda_k(G)$ and $\xi_k(G)$ can be determined by studying the monotonicity of $\xi_k(G)$ as a function on k. For the ease of statement, we say that *vertices in S are black and vertices in $\overline{S}$ are white.*

We shall use induction on the order of the graph to determine $\omega(S)$. To reduce the order, we contract consecutive vertices in $H_{m,d}$. Notice that the contracted graph is no longer a Harary graph. In order that induction hypothesis can be used on the smaller graph, some 'new' edges have to be added. This consideration motivates the following operation on a Harary graph. We define the operation of *collapsing vertex i to vertex j* as follows: suppose i and j are consecutive (in the sense of cyclic order); identify vertex i and vertex j; label the new vertex as j and keep the labels of all the other vertices (thus index i no longer appears in the new graph); then add some edges such that $H_{m-1,2r}$ is embedded in the new graph in the natural way. Vertex i is called the *collapsed vertex.* For example, collapsing vertex i to vertex $i-1$ in the Harary graph $H_{2t,2r+1}$ of the second type (Fig.1a) results in a Harary graph $H_{2t-1,2r+1}$ of the third type (Fig.1b), where new edges $\{(t, t+r+1) : t = i-r, \ldots, i-2\}$ are added in order that $H_{2t-1,2r}$ is embedded in the new graph. Note that the edge $(i, i+r)$ in the original graph collapses onto the edge $(i-1, i+r)$ in the new graph. In order to simplify our statement, we regard all original edges incident with vertex i as *disappeared edges* and all edges in the new graph whose labels (an edge is labeled by the pair of labels of its two ends) are not in the original graph as *new edges.* Hence $(i-1, i+r)$ is regarded as a new edge which is used to replace the disappeared edge $(i, i+r)$. By the same token, the new diagonal $(i-1, i+m/2)$ is used to replace the disappeared diagonal $(i, i+m/2)$. It is easy to see that collapsing two consecutive vertices in a Harary graph $H_{m,2r}$ of the first type results in a smaller Harary graph $H_{m-1,2r}$ of the first type. Collapsing two consecutive vertices i to j in a Harary graph $H_{m,2r+1}$ of the second type results in a smaller Harary graph $H_{m-1,2r+1}$ of the third type; further collapsing $i + \frac{m}{2}$ to $j + \frac{m}{2}$ results in a smaller Harary graph $H_{m-2,2r+1}$ of the second type.

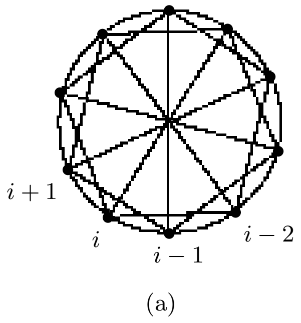
For a Harary graph G and a vertex subset S of $V(G)$, we use G' to denote the new graph obtained from G by one collapse or a sequence of collapses, and use S' to denote the vertex subset of $V(G')$ obtained from S by removing the collapsed vertices (if necessary).

First, we consider changes of non-diagonal edges (that is, edges in $H_{m,2r}$) after one collapse.

Lemma 1. *Let $G = H_{m,2r}$ be a Harary graph of the first type. Suppose i and j are two consecutive vertices, and S is a vertex subset of $V(G)$. Collapse i to j. Then*

$$\omega_G(S) = \omega_{G'}(S') + 2\ell,$$

where $\ell = |\{(a, b) : (a, b)$ is a new edge and the color of i is different from that of a and $b\}|$.



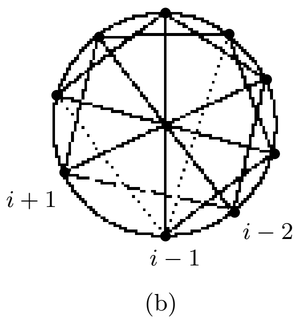


Fig. 1. The dashed line is newly added. The dotted lines are new edges resulted from collapse which are used to take the place of those disappeared edges incident with vertex i in the original graph

Proof. By symmetry, we may assume that $j = i - 1$ and $i \in S$. After collapse, edges incident with vertex i disappear, new edges $\{(t, t{+}r{+}1) \mid t = i{-}r, \ldots, i{-}2\}$ are added, and new edge $(i-1, i+r)$ is used to replace the disappeared edge $(i, i+r)$. For each $t \in \{i-r, ..., i-1\}$, a new edge $(t, t+r+1)$ corresponds to two disappeared edges (t, i) and $(t+r+1, i)$. If exactly one of t and $t+r+1$ has the same color with i, say t, then the edge $(i, t+r+1)$ in $[S, \overline{S}]_G$ disappear and the new edge $(t, t+r+1)$ is in $[S', \overline{S'}]_{G'}$, while the edge (i, t) belongs to neither $[S, \overline{S}]_G$ nor $[S', \overline{S'}]_{G'}$. If the color of i is different from that of t and $t+r+1$, then the two edges (i, t) and $(i, t+r+1)$ in $[S, \overline{S}]_G$ disappear and the new edge $(t, t+r+1)$ belongs to neither $[S, \overline{S}]_G$ nor $[S', \overline{S'}]_{G'}$. If both t and $t+r+1$ have the same color as i, then all the three edges $(i, t), (i, t+r+1)$ and $(t, t+r+1)$ belong to neither $[S, \overline{S}]_G$ nor $[S', \overline{S'}]_{G'}$. Hence only those edges described in the definition of ℓ contribute to $\omega_G(S) - \omega_{G'}(S')$, each contributing 2. The lemma is proved.

Next, we take into account the changes of diagonals in one collapse.

Lemma 2. *Let G be a Harary graph of the second or the third type. Suppose i and j are two consecutive vertices, and S is a vertex subset of $V(G)$. Collapse i to j. Then*

$$\omega_G(S) = \omega_{G'}(S') + 2\ell + p - q + \mu + \nu, \tag{1}$$

where ℓ is as in Lemma 1 and

$p = |\{a_i : (i, a_i)$ is a diagonal of G; j is not adjacent with a_i in G; (j, a_i) is not a new non-diagonal edge in G'; the color of i is different from that of j and $a_i\}|$,

$q = |\{a_i : (i, a_i)$is a diagonal of G; j isnot adjacent with a_i in$G$; (j, a_i) is not a new non-diagonal edge in G'; the color of j is different from that of i and $a_i\}|$,

$\mu = |\{a_i :$ both (i, a_i) and (j, a_i) are diagonals of G; a_i and i have different colors$\}|$,

$\nu = |\{(a, b)$ is a diagonal of $G : (a, b)$ collapses onto a new non-diagonal edge of G'; a, b have different colors$\}|$.

Proof. In view of Lemma 1, we shall concentrate on the changes of diagonals after the collapse. The number of diagonals in $\omega_G(S) - \omega_{G'}(S')$ is affected only by the following three cases.

(1) Edge (i, a_i) is a diagonal of G, vertex j is not adjacent with vertex a_i in G, and (j, a_i) is not a new non-diagonal edge in G'. Then the new diagonal (j, a_i) takes the place of the disappeared diagonal (i, a_i). If the color of i is different from that of j and a_i, then the number of diagonals is decreased by one because the missing diagonal (i, a_i) is in $[S, \overline{S}]_G$ and the new diagonal (j, a_i) is not in $[S', \overline{S'}]_{G'}$. If the color of j is different from that of i and a_i, then the number of diagonals is increased by one because the new diagonal (j, a_i) is in $[S', \overline{S'}]_{G'}$ and the missing diagonal (i, a_i) is not in $[S, \overline{S}]_G$.

(2) Both (i, a_i) and (j, a_i) are diagonals of G. Then, the diagonal (i, a_i) disappears without any replacement. Hence if i and a_i have different colors, then the number of diagonals is decreased by one.

(3) Some diagonal (a, b) of G collapses onto a 'new' non-diagonal edge of G'. Since such an edge has been considered in counting ℓ, it should be viewed as a disappeared edge without any replacement. Hence if a, b have different colors, then the number of diagonals is decreased by one.

Combining Lemma 1 with the above analysis, the lemma is proved.

Corollary 1. *Let $G = H_{m,d}$ be a Harary graph of the second or the third type, S be a vertex subset of $V(G)$ with $|S| \leq m/2$. If there exists a diagonal (i, a_i) such that $i \in S$ and $a_i \in \overline{S}$, then there exist some collapses such that*

$$\omega_G(S) \geq \omega_{G'}(S') + 2\gamma + 1, \tag{2}$$

where $|S| - 1 \leq |S'| \leq |V(G')|/2$, G' is still a Harary graph of the second or the third type, and γ is the sum of some ℓ's as in Lemma 1.

Proof. First, suppose G is of the third type and $i = 0$. In this case, $|S| \leq (m-1)/2$. Let (i, a_i') be the other diagonal incident with i. Collapse a_i to a_i'. Then G' is of the second type, $|S'| = |S| \leq (m-1)/2 = |V(G')|/2$, and inequality (2) follows from Lemma 2 by noting that in this case $q = 0$ and $\mu = 1$.

If G is of the third type and $a_i = 0$, let (j, a_i) be the other diagonal incident with a_i and collapse i to j. Then the result follows from a similar argument as above. The only difference is that in this case $|S'| = |S| - 1 < |V(G')|/2$.

In the above two cases, $\gamma = \ell$.

Next, suppose G is either of the second type or of the third type but $0 \notin \{i, a_i\}$. In this case, $(i-1, a_i-1)$ is a diagonal and $i-1$ is not adjacent with a_i. Collapse i to $i-1$ and a_i to $a_i - 1$. Then G' is of the same type as G, and $|S'| = |S| - 1 \leq |V(G')|/2$. By Lemma 2, we have

$$\omega_G(S) = \omega_{G'}(S') + 2\ell_1 + p_1 - q_1 + \mu_1 + \nu_1 + 2\ell_2 + p_2 - q_2 + \mu_2 + \nu_2, \tag{3}$$

where p_1, q_1, μ_1, ν_1 are the parameters corresponding to the collapse of i to $i-1$, and p_2, q_2, μ_2, ν_2 correspond to the collapse of a_i to $a_i - 1$. By the definition of q and the assumption that i and a_i are of different colors, we have $q_1 = q_2 = 0$.

Furthermore, if $i-1 \in \overline{S}$ and $(i-1, a_i)$ is not a new non-diagonal edge, then $p_1 \geq 1$; if $i-1 \in \overline{S}$ and $(i-1, a_i)$ is a new non-diagonal edge, then $\nu_1 \geq 1$; if $i-1 \in S$, then $\mu_2 \geq 1$. By setting $\gamma = \ell_1 + \ell_2$, inequality (2) follows from (3).

Next, we consider changes of non-diagonal edges after collapsing three consecutive vertices.

Lemma 3. *Let $G = H_{m,2r}$ be a Harary graph of the first type, $i-1, i, i+1$ be three consecutive vertices of G. If the color of $i-1$ is different from that of i and $i+1$, then collapsing both $i-1$ and $i+1$ to i results in $\omega_G(S) = \omega_{G'}(S') + 2\gamma$ for some $\gamma \geq 0$. Furthermore, $\gamma = 0$ if and only if all vertices $i-1, i-2, \ldots, i-r-1$ are of the same color and all vertices $i, i+1, \ldots, i+r \in \overline{S}$ are of the other color.*

Proof. For simplicity of statement, we assume that $i-1 \in S$ and $i, i+1 \in \overline{S}$. Collapsing $i-1$ and $i+1$ to i can be carried out in two steps: First, collapse $i-1$ to i and let G_1, S_1 be the resulting graph and vertex subset respectively. Then, collapse $i+1$ to i, resulting in G' and S'. By Lemma 1, $\omega_G(S) = \omega_{G_1}(S_1) + 2\ell_1 = \omega_{G'}(S') + 2(\ell_1 + \ell_2) = \omega_{G'}(S') + 2\gamma$, where ℓ_1 and ℓ_2 are the ℓ's as in Lemma 1 corresponding to the two collapses respectively.

Clearly, $\gamma = 0$ if and only if $\ell_1 = \ell_2 = 0$. In the first collapse, the set of new edges is $E_1 = \{(t, t-r-1) : t = i, i+1, ..., i+r-1\}$. Since $i-1 \in S$, $i, i+1 \in \overline{S}$, and $(i, i-r-1), (i+1, i-r) \in E_1$, it follows from $\ell_1 = 0$ and the definition of ℓ that $i-r-1, i-r \in S$. In the second collapse, the set of new edges is $E_2 = \{(i, i+r+1)\} \cup \{(t, t+r+2) : t = i-2, i-3, ..., i-r\}$. Since $i+1 \in \overline{S}$, $i-r \in S$, and $(i-r, i+2) \in E_2$, it follows from $\ell_2 = 0$ that $i+2 \in \overline{S}$. Since $(i+2, i-r+1) \in E_1$, we obtain $i-r+1 \in S$ by $\ell_1 = 0$. Since $(i-r+1, i+3) \in E_2$, we have $i+3 \in \overline{S}$ by $\ell_2 = 0$. Proceeding like this, the second half of the lemma is proved.

3 Harary Graph of the First Type

In this section, we prove the λ_k-optimality of Harary graphs of the first type.

Lemma 4. *Let $G = H_{m,2r}$ be a Harary graph of the first type, S be a vertex subset with $|S| = k$, where $r \leq k \leq \frac{m}{2}$. Then $\omega(S) \geq r(r+1)$.*

Proof. When $k = r$, we have $\omega(S) = 2rk - 2|E(G[S])| \geq 2rk - k(k-1) = r(r+1)$ (equality holds if and only if $G[S]$ is a complete subgraph). In the following, we prove the lemma by induction on m. Since G is $2r$-regular, we have $m \geq 2r+1$. If $m = 2r+1$, then by $r \leq k \leq m/2$, we have $k = r$, and the result is true. Next, suppose $m \geq 2r+2$, $k \geq r+1$, and the result is true for any Harary graph of the first type with fewer vertices. Let i be a vertex in S and collapse i to $i-1$. Since $|S'| = |S| - 1 = k-1$ satisfies the induction hypothesis that $r \leq |S'| \leq (m-1)/2$, it follows from Lemma 1 that $\omega_G(S) \geq \omega_{G'}(S') \geq r(r+1)$.

Theorem 1. *Let $G = H_{m,2r}$ be a Harary graph of the first type. Then G is λ_k-optimal for any $1 \leq k \leq \frac{m}{2}$. Furthermore,*

$$\lambda_k(G) = \begin{cases} 2rk - k(k-1), & \text{when } k < r, \\ r(r+1), & \text{when } r \leq k \leq \frac{m}{2}. \end{cases}$$

Proof. When $k \leq r$, a consecutive section of k vertices induces a complete subgraph of G, and thus achieves the minimum of $\omega(S)$ among all k-subsets S (since G is regular). Hence $\xi_k(G) = 2rk - k(k-1)$ for $k \leq r$. When $r \leq k \leq m/2$, $\omega(S) \geq r(r+1)$ by Lemma 4. Since a consecutive section of k vertices achieves this lower bound, we have $\xi_k(G) = r(r+1)$ for $r \leq k \leq m/2$.

Note that a λ_k-cut divides the graph into exactly two connected components. Hence $\lambda_k(G)$ can be re-expressed as

$$\lambda_k(G) = \min\{\omega(S) : |S| \geq k, |\overline{S}| \geq k, G[S] \text{ and } G[\overline{S}] \text{ are both connected}\}.$$

Note that for each $1 \leq k \leq m/2$, there exists a k-subset S of $V(G)$ such that $\omega(S) = \xi_k(G)$ and both $G[S]$ and $G[\overline{S}]$ are connected (any consecutive section of k vertices may serve as such an S). Hence $\lambda_k(G) = \min\{\xi_j(G) : k \leq j \leq m/2\}$. Then the theorem follows from the observation that $\xi_k(G)$ is a monotonely non-decreasing function on k.

4 Harary Graphs of the Second or the Third Type

The proof of Theorem 1 illustrates the main idea of our proofs: Compare $\omega_G(S)$ and $\omega_{G'}(S')$, then use induction hypothesis on $\omega_{G'}(S')$ to find out a tight lower bound for $\omega_G(S)$.

The following lemma considers a special distribution of S.

Lemma 5. *Let $G = H_{m,d}$ be a Harary graph of the second type with $m = 4L$ for some integer $L \geq 2$. Suppose $d = 2r+1$, $S = \{0, 2, 4, \cdots, m-4, m-2\}$, and $k = \frac{m}{2}$.*

(i) If $k = r+2$, then $\omega(S) = r(r+1) + k - 2$;

(ii) If $k > r+2$, then $\omega(S) \geq r(r+1) + k$.

Proof. In this case, $|S| = k = \frac{m}{2} = 2L$ is an even number, and $[S, \overline{S}]$ does not contain diagonals. For each vertex i, there are $2\lfloor\frac{r+1}{2}\rfloor$ edges in $[S, \overline{S}]$ incident with i. It follows that $\omega(S) = 2k\lfloor\frac{r+1}{2}\rfloor$. If r is odd, then $\omega(S) = k(r+1) = kr + k \geq r(r+2) + k > r(r+1) + k$. Next, suppose r is even. Then $r \geq 2$. If $k = r+2$, then $\omega(S) = kr = r(r+1) + k - 2$. If $k \geq r+3$, then $k \geq r+4$ since both k and r are even. Thus $\omega(S) = kr = k(r-1) + k \geq (r+4)(r-1) + k = r(r+1) + 2(r-2) + k \geq r(r+1) + k$. The proof is completed.

Lemma 6. *Let G be a Harary graph of the second or the third type, k be an integer with $r \leq k \leq \min\{r^2 + r, \frac{m}{2}\}$, S be a k-subset of $V(G)$. Then, except for the case in Lemma 5 (i),*

$$\omega(S) \geq r(r+1) + k.$$

Proof. The result is true if G is in the case of Lemma 5 (ii). Hence in the following, we assume that G is not in the case of Lemma 5.

We prove the lemma by induction on k. Since G has minimum degree $d = 2r + 1$, we have

$$\omega(S) = \sum_{v \in S} d(v) - 2|E(G[S])| \geq k \cdot d - k(k-1) = k(2r + 1 - k) + k.$$

Hence when $k = r$ or $r + 1$, the result is true. For the induction step, suppose $k \geq r + 2$ and the result holds for any integers smaller than k. We consider two cases:

Case 1. There exists a diagonal (i, a_i) of G such that $i \in S$ and $a_i \in \overline{S}$.

By Corollary 1, there exist some collapses such that G' is of the second or the third type, S' satisfies $r + 1 \leq k - 1 = |S| - 1 \leq |S'| \leq |V(G')|/2$, and

$$\omega_G(S) \geq \omega_{G'}(S') + 2\gamma + 1.$$

If G' and S' are not in the case of Lemma 5 (i), then by induction hypothesis,

$$\omega_G(S) \geq r(r+1) + |S'| + 2\gamma + 1 \geq r(r+1) + k.$$

If G' and S' are in the case of Lemma 5 (i), then

$$\omega_G(S) \geq r(r+1) + |S'| - 2 + 2\gamma + 1 \geq r(r+1) + k + 2(\gamma - 1).$$

In this case, $r = |S'| - 2$ is even. Hence for each new non-diagonal edge (a, b) in G', since a and b are at distance r along the outer circle of G', we see from the distribution of S' that a, b are of the same color. Furthermore, in the last collapse of the sequence of collapses as described in the proof of Corollary 1, which creates r new non-diagonal edges, $r/2$ of them have their ends having different color from the collapsed vertex. Hence it follows from $r \geq 2$ that $\gamma \geq 1$. The result follows.

Case 2. Every diagonal of G has its two ends having the same color.

In this case, there is no diagonal in $[S, \overline{S}]_G$. If m is odd, then G is of the third type, and $|S| \leq \lfloor m/2 \rfloor = (|V(G)| - 1)/2$. It follows that there exist two consecutive vertices in $\overline{S}$. If m is even, then G is of the second type. Since we have assumed that the case in Lemma 5 does not occur, the existence of two consecutive vertices in $\overline{S}$ also holds. Hence we may assume that there are three consecutive vertices $i-1, i, i+1$ such that $i - 1 \in S$ and $i, i+1 \in \overline{S}$. Let $(i-1, a_{i-1}), (i, a_i), (i+1, a_{i+1})$ be three diagonals such that a_{i-1}, a_i, a_{i+1} are consecutive. By the assumption of Case 2, $a_{i-1} \in S$ and $a_i, a_{i+1} \in \overline{S}$. Hence a_{i+1} may coincide with a_i but a_{i-1} is different from a_i and a_{i+1}. Collapse $i-1, i+1$ to i and a_{i-1} to a_i. If $a_{i+1} \neq a_i$, also collapse a_{i+1} to a_i.

First, we suppose G', S' are not in the case of Lemma 5 (i).

When either (a) G is of the second type, or (b) G is of the third type and $a_{i+1} = a_i$, or (c) G is of the third type, $a_{i+1} \neq a_i$, and $0 \notin \{i-1, a_{i-1}\}$, we see that the number of diagonals in $\omega_G(S) - \omega_{G'}(S')$ does not change. Since $|S'| = k - 2 = |S| - 2$ and $r + 2 \leq k \leq m/2$, we have $r \leq |S'| \leq m/2 - 2 \leq |V(G')|/2$. Hence by Lemma 3 and Lemma 1, we have

$$\omega_G(S) = \omega_{G'}(S') + 2(\gamma_1 + \gamma_2), \tag{4}$$

where γ_1, γ_2 result from collapsing $i-1, i+1$ to i and a_{i-1}, a_{i+1} to a_i (or a_{i-1} to a_i if $a_i = a_{i+1}$) respectively. Then by induction hypothesis,

$$\omega_G(S) \geq r(r+1) + |S'| + 2(\gamma_1 + \gamma_2) \geq r(r+1) + k + 2(\gamma_1 + \gamma_2 - 1).$$

If $\gamma_1 \geq 1$, then we are done. If $\gamma_1 = 0$, then by Lemma 3,

$$i-1, i-2, \ldots, i-r-1 \in S \text{ and } i, i+1 \ldots, i+r \in \overline{S}. \tag{5}$$

When G is of the third type, $a_{i+1} \neq a_i$, and $0 \in \{i-1, a_{i-1}\}$ (say $i-1=0$ by symmetry), let $(i-1, a'_{i-1})$ be the other diagonal incident with $i-1$. By the assumption of Case 2, $a'_{i-1} \in S$, and thus $a'_{i-1} = a_{i-2}$. Similar to the above analysis, by noting that the number of diagonals in $\omega_G(S) - \omega_{G'}(S')$ is decreased by one if (i, a_{i-2}) is not a new non-diagonal edge after collapsing $i-1$ to i (since in this case (i, a_{i-2}) is a new diagonal in $[S', \overline{S'}]_{G'} \setminus [S, \overline{S}]_G$), we have

$$\omega_G(S) \geq \omega_{G'}(S') - 1 + 2(\gamma_1 + \gamma_2). \tag{6}$$

Then by induction hypothesis,

$$\omega_G(S) \geq r(r+1) + k + 2(\gamma_1 + \gamma_2 - 1) - 1.$$

Recall that γ is the sum of some ℓ's. By the definition of ℓ, the hypothesis of Case 2, and the assumption at the beginning of this paragraph, we have $\gamma_1 = \gamma_2$. If $\gamma_1 = \gamma_2 \geq 1$, then we are done. Otherwise, $\gamma_1 = 0$, and the distribution (5) also holds by Lemma 3.

In view of the above analysis, with symmetry in mind, we may assume that for any three consecutive vertices $i-1, i, i+1$ with $i, i+1$ having different color from $i-1$, all vertices $i-1, i-2, \ldots, i-r-1$ are of the same color and all vertices $i, i+1, \ldots, i+r$ are of the other color. Thus the distribution of vertices in S are such that each consecutive section of S has at least $r+1$ vertices and the gap between any two consecutive sections of S has at least $r+1$ vertices of $\overline{S}$. Combining this with the hypothesis of Case 2, we may assume that S consists of $2p$ consecutive sections for some $p \geq 1$. Since each consecutive section R of S in such a configuration has $[R, \overline{S}]_G = r(r+1)$, we have $\omega_G(S) = 2p \cdot r(r+1) \geq 2r(r+1) \geq r(r+1) + k$.

Next, Suppose G' and S' are in the case of Lemma 5 (i). Similar to the above deduction, we still have inequalities (4) and (6), but the induction hypothesis leads to

$$\omega_G(S) \geq r(r+1) + k + 2(\gamma_1 + \gamma_2 - 2) - 1.$$

Similar to the analysis in Case 1, each new non-diagonal edge of G' has its ends having the same color. Since $(i, i-r-1)$ is a new non-diagonal edge whose ends are in $\overline{S}$ which have different color from the collapsed vertex $i-1$, $(i+2, i-r)$ is a new non-diagonal edge whose ends are in S which have different color from the collapsed vertex $i+1$, and $(a_i, a_i - r - 1)$ is a new non-diagonal edge whose ends are in $\overline{S}$ which have different color from the collapsed vertex a_{i-1}, we see that $\gamma_1 + \gamma_2 \geq 3$ and thus $\omega_G(S) > r(r+1) + k$.

The lemma is proved.

Lemma 7. *Let $G = H_{m,d}$ be a Harary graph of the second or the third type, k be an integer with $r^2 + r \le k \le \frac{m}{2}$, S be a k-subset of $V(G)$. Then*

$$\omega(S) \ge \begin{cases} 2r(r+1)+1, & \text{when } k \text{ is odd and } G \text{ is of the second type;} \\ 2r(r+1), & \text{otherwise.} \end{cases}$$

Proof. First, we see that $r^2 + r \le k \le m/2$ ensures that G is not in the case of Lemma 5 (i). Otherwise, $r+2 = k \ge r^2+r$ implies that $r = 1$ and $k = 3$. But then $m = 2k = 6$ is not divisible by four. In the following, all the collapsed graph G' and the corresponding vertex subset S' satisfy $r^2 + r \le |S'| \le |V(G')|/2$, hence are not in the case of Lemma 5 (i) too.

We prove the lemma by induction on m. By $r^2 + r \le k \le m/2$, we have $m \ge 2r(r+1)$. If $m = 2r(r+1)$ or $2r(r+1)+1$, then $k = r(r+1)$ is even. By Lemma 6, we have $\omega(S) \ge r(r+1) + k = 2r(r+1)$.

For the induction step, suppose $k \ge r(r+1)+1$ and the result is true for any Harary graph of the second or the third type with fewer vertices. If there is a diagonal (i, a_i) with $i \in S$ and $a_i \in \overline{S}$, then by Corollary 1 and the induction hypothesis, we have $\omega_G(S) \ge \omega_{G'}(S') + 1 \ge 2r(r+1)+1$.

Next, assume that

$$\text{each diagonal of } G \text{ has its two ends having the same color.} \tag{7}$$

If G is of the second type, then k is even by assumption (7), and thus $k \ge r(r+1)+1$ implies that $k \ge r(r+1)+2$. If $k \le m/2 - 1$, then there exist two consecutive vertices $i-1$ and i both in $\overline{S}$. Collapse i to $i-1$. By Lemma 2, noting that $i, a_i, i-1 \in \overline{S}$ implies $q = 0$, and $r(r+1) < |S'| = |S| < |V(G')|/2$ satisfies the induction hypothesis, we have $\omega_G(S) \ge \omega_{G'}(S) \ge 2r(r+1)$. Suppose $k = m/2$. If there exist two consecutive vertices $i-1$ and i both in S, we collapse i to $i-1$. Then the desired $\omega_G(S) \ge 2r(r+1)$ follows from a similar argument as the above, noting that in this case $i, a_i, i-1 \in S$ implies $q = 0$, and $r(r+1) \le |S'| = |S| - 1 < |V(G')|/2$ satisfies the induction hypothesis. Hence suppose that no vertices in S are adjacent. Since $|S| = k = m/2$, vertices in S and $\overline{S}$ appear on the circle alternately. By symmetry, we may assume that $S = \{0, 2, 4, ..., m-2\}$. Since k is even, m is divisible by four. Hence we are in the case of Lemma 5, and thus $\omega(S) \ge r(r+1) + k - 2 \ge 2r(r+1)$.

If G is of the third type, then $k \le \frac{m-1}{2}$. Collapse $\frac{m-1}{2}$ to $\frac{m+1}{2}$. Then $r(r+1) \le |S| - 1 \le |S'| \le |S| \le (m-1)/2 = |V(G')|/2$. The result follows form Lemma 2 and the induction hypothesis, by noting that $q = 0$.

Lemma 8. *Let $G = H_{m,d}$ be a Harary graph. If G is of the third type, then*

$$\xi_k(G) = \begin{cases} (2r+1)k - k(k-1), & \text{when } k \le r+1, \\ r(r+1)+k, & \text{when } r+2 \le k \le \min\{r^2+r, \frac{m}{2}\}, \\ 2r(r+1), & \text{when } r^2+r+1 \le k \le \frac{m}{2}; \end{cases}$$

if G is of the second type, then

$$\xi_k(G) = \begin{cases} (2r+1)k - k(k-1), & \textit{when } k \le r+1, \\ r(r+1)+r, & \textit{when } k = r+2 = 2L \textit{ and } m = 4L \\ & \textit{for some } L \ge 2, \\ r(r+1)+k, & \textit{when } r+3 \le k \le \min\{r^2+r, \frac{m}{2}\} \\ & \textit{or } k = r+2 \textit{ and } [m \textit{ is not divisible} \\ & \textit{by } 4 \textit{ or } k \ne \frac{m}{2}], \\ 2r(r+1), & \textit{when } r^2+r+1 \le k \le \frac{m}{2} \textit{ and } k \textit{ is even}, \\ 2r(r+1)+1, & \textit{when } r^2+r+1 \le k \le \frac{m}{2} \textit{ and } k \textit{ is odd}. \end{cases}$$

Furthermore, for each $1 \le k \le m/2$, there exists a subset S of k vertices such that $\omega(S) = \xi_k(G)$ and $G[S], G[\overline{S}]$ are both connected.

Proof. When $k \le r+1$, a consecutive section of $\{0, 1, \dots, m-1\}$ avoiding vertex 0 induces a complete subgraph of $H_{m,d}$, and thus achieves the minimum of $\omega(S)$ among all k-subsets S. Hence $\xi_k(G) = (2r+1)k - k(k-1)$. In the following, suppose $k \ge r+1$.

First, suppose G is of the third type. When $r+1 \le k \le r^2+r$, we have $\omega(S) \ge r(r+1)+k$ by Lemma 6. Since a consecutive section of $\{0, 1, \dots, m-1\}$ avoiding vertex 0 achieves this lower bound, we have $\xi_k(G) = r(r+1)+k$. When $r^2+r \le k \le \frac{m}{2}$, by Lemma 7, we have $\omega(S) \ge 2r(r+1)$. Let $S_0 = \{0, \dots, \frac{k-3}{2}\} \cup \{\frac{m-1}{2}, \dots, \frac{m+k-2}{2}\}$ when k is odd and $S_0 = \{1, \dots, \frac{k}{2}\} \cup \{\frac{m+3}{2}, \dots, \frac{m+k+1}{2}\}$ when k is even. Then $|S_0| = k$ and $\omega(S_0)$ achieves the above lower bound. Hence $\xi_k(G) = 2r(r+1)$.

Next, suppose G is of the second type. When $m = 4L$ for some $L \ge 2$ and $k = r+2 = m/2$, if $S = \{0, 2, 4, \cdots, m-2\}$, then $\omega(S) = r(r+1)+k-2 = r(r+1)+r$ by Lemma 5 (i); if $S \ne \{0, 2, 4, \cdots, m-2\}$, then by Lemma 6 or Lemma 7, either $\omega(S) \ge r(r+1)+k > r(r+1)+r$ or $\omega(S) \ge 2r(r+1) > r(r+1)+r$. Since $r = k-2 = 2L-2$ is even, we have $r \ge 2$. Hence $S_0 = \{0, 2, \dots, m-2\}$ induces a connected subgraph of G such that $G[\overline{S_0}]$ is also connected and $\omega(S_0) = kr = r(r+2)$ achieves the lower bound $r(r+1)+r$. It follows that $\xi_k(G) = r(r+1)+r$.

When $r+1 \le k \le r^2+r$ and m, k are not in the above case, we have $\omega(S) \ge r(r+1)+k$ by Lemma 6. Since a consecutive section of $\{0, 1, ..., m-1\}$ achieves this lower bound, we have $\xi_k(G) = r(r+1)+k$.

When $r^2+r+1 \le k \le \frac{m}{2}$, by Lemma 7, we have $\omega(S) \ge 2r(r+1)$ for even k, and $\omega(S) \ge 2r(r+1)+1$ for odd k. Let S_0 be a k-subset of $\{0, 1, \dots, m-1\}$ consisting of two 'nearly' antipodal consecutive segments, that is, $S_0 = \{0, \dots, \frac{k}{2}-1\} \cup \{\frac{m}{2}, \dots, \frac{m}{2}+\frac{k}{2}-1\}$ when k is even and $S_0 = \{0, \dots, \frac{k-3}{2}\} \cup \{\frac{m}{2}, \dots, \frac{m}{2}+\frac{k-1}{2}\}$ when k is odd. Then $\omega(S_0)$ achieves the above lower bound. Hence $\xi_k(G) = 2r(r+1)$ for even k and $\xi_k(G) = 2r(r+1)+1$ for odd k.

Theorem 2. *Let $G = H_{m,d}$ be a Harary graph of the second or the third type. For any integer $1 \le k \le m/2$, G is λ_k-optimal except when G is of the second type and*

(1) either $m \geq 8$ is divisible by 4, $r = m/2 - 2$, and $k = r + 1$, or

(2) $r^2 + r + 1 \leq k < m/2$ and k is odd.

Furthermore, in the case that G is of the second type, $m \geq 8$ is divisible by 4, and $r = m/2 - 2$,

$$\lambda_k(G) = \begin{cases} (2r+1)k - k(k-1), & \textit{when } k \leq r, \\ r(r+1) + r, & \textit{when } k = r+1, r+2. \end{cases}$$

Except for the above case,

$$\lambda_k(G) = \begin{cases} (2r+1)k - k(k-1), & \textit{when } k \leq r+1, \\ r(r+1) + k, & \textit{when } r+2 \leq k \leq \min\{r^2 + r, \frac{m}{2}\}, \\ 2r(r+1), & \textit{when } r^2 + r + 1 \leq k \leq \frac{m}{2}. \end{cases}$$

Proof. The proof is similar to that in Theorem 1, except that $\xi_k(G)$ is not always monotonely non-decreasing when G is of the second type and one of the following two cases occurs:

(1) $m \geq 8$ is divisible by 4 and $r = m/2 - 2$. In this case, $\xi_k(G)$ strictly increases up to $k \leq r + 1$ and $\xi_{r+2}(G) = r(r+1) + r = \xi_{r+1}(G) - 1$.

(2) $r^2 + r + 1 \leq m/2 - 2$. In this case, $\xi_k(G)$ strictly increases up to $k \leq r^2 + r + 2$. After that, for $k = r^2 + r + 3, r^2 + r + 4, ..., m/2$, the values of $\xi_k(G)$ alternate between $2r(r+1)$ and $2r(r+1) + 1$.

The remaining proofs are exactly the same as that in Theorem 1.

5 Concluding Remark

In this paper, we show that except for two cases, the Harary graph is λ_k-optimal for any $k \leq |V(G)|/2$. In recent years, there are a lot of studies on λ_k-optimal graphs for $k = 2, 3$, while the studies for general k are relatively less. The general rules for general k need to be explored further.

An important problem in network design is as follows: given the number of vertices and the number of edges, how to find a graph which are optimal with respect to some criteria? Notice that in the three types of Harary graphs, which are known as *elementary Harary graphs*, the number of edges are determined by the number of vertices. Hence the results in this paper is not a complete solution to the above problem with λ_k-optimality being the criteria. A possible way might be to consider the *generalized Harary graph* which is obtained from the elementary Harary graph by adding more edges while keeping the minimum degree.

References

1. Balbuena, C., Carmona, A., Fàbrega, J., Fiol, M.A.: Extraconnectivity of graphs with large minimum degree and girth. Discrete Mathematics 167/168, 85–100 (1997)

2. Bauer, D., Boesch, F., Suffel, C., Van Slyke, R.: On the validity of a reduction of reliable network design to a graph extremal problem. IEEE Tran. Circuits and Systems 34, 1579–1581 (1989)
3. Bondy, J.A., Murty, U.S.R.: Graph Theory with Application. Macmillan, London (1976)
4. Bonsma, P., Ueffing, N., Volkmann, L.: Edge-cuts leaving components of order at least three. Discrete Math. 256, 431–439 (2002)
5. Deng, H., Chen, J., Li, Q., Li, R., Gao, Q.: On the construction of most reliable networks. Discrete Appl. Math. 140, 19–33 (2004)
6. Esfahanian, A.H., Hakimi, S.L.: On computing a conditional edge connectivity of a graph. Inform. Process. Lett. 27, 195–199 (1988)
7. Fàbrega, J., Fiol, M.A.: Extraconnectivity of graphs with large girth. Discrete Math. 127, 163–170 (1994)
8. Harary, F.: The maximum connectivity of a graph. Proc. Nat. Acad. Sci. U.S.A. 48, 1142–1146 (1962)
9. Li, Q.L., Li, Q.: Reliability analysis of circulant graphs. Networks 31, 61–65 (1998)
10. Hellwig, A., Volkmann, L.: Maximally edge-connected and vertex-connected graphs and digraphs: A survey. Discrete Math. 308, 3265–3296 (2008)
11. Meng, J.X.: Optimally super-edge-connected transitive graphs. Discrete Math. 260, 239–248 (2003)
12. Meng, J.X., Ji, Y.H.: On a kind of restricted edge connectivity of graphs. Discrete Applied Math. 117, 183–193 (2002)
13. Ou, J.P.: Edge cuts leaving components of order at least m. Discrete Math. 305, 365–371 (2005)
14. Wang, G., Zhang, L.: The structure of maxλ-min$m_{\lambda+1}$ graphs used in the design of reliable networks. Networks 30, 231–242 (1997)
15. Wang, M., Li, Q.: Conditional edge connectivity properties, reliability comparison and transitivity of graphs. Discrete Math. 258, 205–214 (2002)
16. Wang, S.Y., Lin, S.W., Li, C.F.: Sufficient conditions for super k-restricted edge connectivity in graphs of diameter 2. Discrete Mathematics 309, 908–919 (2009)
17. Xu, J.M., Xu, K.L.: On restricted edge-connectivity of graphs. Discrete Math. 243, 291–298 (2002)
18. Zhang, Z., Yuan, J.J.: Degree conditions for restricted-edge-connectivity and iosperimetric-edge-connecitivity to be optimal. Discrete Math. 307, 293–298 (2007)
19. Zhang, Z., Yuan, J.J.: A proof of an inequality concerning k-restricted edge connectivity. Discrete Math. 304, 128–134 (2005)

Efficient Algorithms for Finding the k Most Vital Edges for the Minimum Spanning Tree Problem

Cristina Bazgan, Sonia Toubaline, and Daniel Vanderpooten

Université Paris-Dauphine, LAMSADE,
Place du Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France
`{bazgan,toubaline,vdp}@lamsade.dauphine.fr`

Abstract. We study in this paper the problem of finding in a graph a subset of k edges whose deletion causes the largest increase in the weight of a minimum spanning tree. We propose for this problem an explicit enumeration algorithm whose complexity, when compared to the current best algorithm, is better for general k but very slightly worse for fixed k. More interestingly, unlike in the previous algorithms, we can easily adapt our algorithm so as to transform it into an implicit exploration algorithm based on a branch and bound scheme. We also propose a mixed integer programming formulation for this problem. Computational results show a clear superiority of the implicit enumeration algorithm both over the explicit enumeration algorithm and the mixed integer program.

Keywords: most vital edges, minimum spanning tree, exact algorithms, mixed integer program.

1 Introduction

In many applications involving the use of communication or transportation networks, we often need to identify critical infrastructures. By critical infrastructure we mean a set of links whose damage causes the largest perturbation within the network. Modeling this network by a weighted graph, identifying critical infrastructures amounts to finding a subset of edges whose removal from the graph causes the largest increase in the cost. In the literature this problem is referred to as the k most vital edges problem. In this paper, we are interested in determining a subset of edges of the graph whose deletion causes the largest increase in the weight of a minimum spanning tree (MST). This problem is referred to as k Most Vital Edges MST.

The problem of finding the k most vital edges of a graph has been studied for various problems including shortest path [1,7,11] and maximum flow [18,14,19]. For the minimum spanning tree problem defined on a graph G with n vertices and m edges, Frederickson *et al.* [4] showed that, for general k, k Most Vital Edges MST is *NP*-hard and proposed an $O(\log k)$-approximation algorithm. For a fixed k the problem is obviously polynomial. The case $k = 1$ has been largely studied in the literature [5,6,16]. Hsu *et al.* [5] gave two algorithms in $O(m \log m)$ and

$O(n^2)$. Iwano and Katoh [6] proposed an algorithm in $O(m\alpha(m,n))$ using Tarjan's result [17], where α is the inverse-Ackermann function. Pettie [12] improved the results of Tarjan[17] and Dixon *et al.* [3], and therefore the current best deterministic algorithm for solving the case $k = 1$ is in $O(m \log \alpha(m,n))$. Several exact algorithms based on an explicit enumeration of possible solutions have been proposed [8,9,15]. The best one [8] runs in time $O(n^k \alpha((k+1)(n-1),n))$ and was achieved by reducing G to a sparse graph. Using Pettie's result [12], the running time of the later algorithm becomes $O(n^k \log \alpha((k+1)(n-1),n))$.

In this paper we propose a new efficient algorithm also based on an explicit enumeration of all possible solutions for k Most Vital Edges MST. Its complexity $O(n^k \log \alpha(2(n-1),n))$ for fixed k is theoretically very slightly worse than the complexity of the algorithm proposed by Liang [8] using Pettie's result [12]. However, given the fact that $\alpha(m,n)$ is always less than 4 in practice, the complexity of these two algorithms can be deemed as equivalent. Moreover, the complexity of our algorithm is better than that of Liang's algorithm for general k. More interestingly, unlike any other algorithm, our algorithm has two specific useful features. First, it can also determine an optimal solution for i Most Vital Edges MST, for each $1 \leq i \leq k$, with the same time complexity. Second, it can be easily adapted to establish an implicit enumeration algorithm based on a branch and bound procedure. We also present in this paper a formulation by a mixed integer program to solve k Most Vital Edges MST. We implement and test all these proposed algorithms using, for the implicit enumeration algorithm, different branching and evaluation strategies. The results show that the implicit enumeration algorithm is much faster than the explicit enumeration algorithm as well as the resolution of the mixed integer program and its use of memory space can handle instances of significantly larger size. Moreover, we propose an ε-approximate algorithm.

The rest of the paper is organized as follows. In section 2 we introduce notations and some results related to our problem. In section 3 we present a new explicit enumeration algorithm that solves k Most Vital Edges MST. In section 4 we propose another exact algorithm based on an implicit enumeration scheme. In section 5, we present a mixed integer programming formulation for k Most Vital Edges MST. Computational results are presented in section 6. In section 7, we present an ϵ-approximate algorithm and compare it with the exact one. Conclusions are provided in section 8.

2 Basic Concepts and Preliminary Results

Let $G = (V,E)$ be a weighted undirected connected graph with $|V| = n$, $|E| = m$ and $w(e) \geq 0$ is the integer weight of each edge $e \in E$. We denote by $G - E'$ the graph obtained from G by removing the subset of edges $E' \subseteq E$. k Most Vital Edges MST consists of finding a subset of edges $S^* \subseteq E$ with $|S^*| = k$ that maximizes the weight of a MST in the graph $G - S^*$. We assume that G is at least $(k+1)$ edge-connected, since otherwise any selection of k edges including the edges of a minimum unweighted cut is a trivial solution. Therefore, we assume $k \leq \lambda(G)-1$, where $\lambda(G)$ is the edge-connectivity of G. Also, without loss of generality, we suppose in the following that all weights are different (by introducing, if necessary, an arbitrary total order on edges with the same weight). This assumption implies the uniqueness of minimum spanning trees or forests. For a non necessarily connected graph, a minimum spanning forest (MSF) is the union of minimum spanning trees for each of its connected components. In this paper a tree or a forest is considered as a graph but also, for convenience, as a subset of edges. For a set of edges F, $w(F)$ represents the sum of the weights of the edges in F.

We denote by T_0 the MST of G. Remark that an optimal solution of k Most Vital Edges MST must contain at least one edge of T_0. For $i \geq 1$, let T_i be the MSF of the graph $G_i = G - \cup_{j=0}^{i-1} T_j$. We use in the following the graph $U_k^G = (V, \cup_{j=0}^{k} T_j)$ which has the following interesting property.

Lemma 1. *(Liang and Shen [9]) For any $S \subseteq E$, $|S| \leq k$, any edge of the MST of graph $G - S$ belongs to U_k^G.*

By Lemma 1, solving k Most Vital Edges MST on G reduces to solving the same problem on the sparser graph U_k^G whose number of edges is at most $(k+1)(n-1)$.

Considering T a MST of a graph, the replacement edge $r(e)$ for an edge $e \in T$ is defined as the edge $e' \neq e$ of minimum weight which connects the two disconnected components of $T \setminus \{e\}$. The sensitivity of a minimum spanning tree T, i.e. the allowable variation for each edge weight so that T remains a minimum spanning tree, can be computed in $O(m \log \alpha(m,n))$ [12]. In particular, for edges in T, this algorithm provides replacement edges. As a consequence, we get the following result.

Lemma 2. 1 Most Vital Edges MST *defined on a graph with n vertices and m edges is solvable in $O(m \log \alpha(m,n))$.*

Proof: Let T^* be a minimum spanning tree in a given graph. We calculate the replacement edges $r(e)$ for all edges $e \in T^*$. The most vital edge is the edge e^* such that $w(r(e^*)) - w(e^*) = \max_{e \in T^*} w(r(e)) - w(e)$. □

Actually, replacement edges belong to a specific subset of edges as shown by the following result.

Lemma 3. *For each edge $e \in T_i$, we have $r(e) \in T_{i+1}$ for $i = 0, \ldots, k-1$.*

Proof: Given a graph G, Liang [8] shows that for each edge $e \in T_0$, $r(e) \in T_1$. Applying this to graph G_i, for which T_i is the MSF, we get the result. □

3 An Explicit Enumeration Algorithm for Finding the k Most Vital Edges

We propose an algorithm that constructs a search tree of depth $k-1$ in a breadth-first mode. At the i^{th} level of this search tree, $i = 0, \ldots, k-1$, a node s is characterized by:

- $mv(s)$: a subset of i edges, corresponding to a tentative partial selection of the k most vital edges.
- $\widetilde{U}(s) = U^{G'(s)}_{k-|mv(s)|}$ where $G'(s) = (V, E \backslash mv(s))$. Hence, $\widetilde{U}(s) = (V, \cup_{i=0}^{k-|mv(s)|} T_i(s))$ where $T_i(s)$ is the MSF in $G'(s) - \cup_{j=0}^{i-1} T_j(s)$.
- $mst(s)$: a subset of edges forbidden to deletion. These edges belonging to $T_0(s)$, will necessary belong to any MST associated with any descendant of s. Depending on the position of s in the search tree, the cardinality of $mst(s)$ varies from 0 to $n-2$.

Denote by N_i, for $i = 0, \ldots, k-1$, the set of nodes of the search tree at the i^{th} level. We describe in the following the exact algorithm.

We first construct the graph U^G_k. Let a be the root of the search tree with $mv(a) = mst(a) = \emptyset$, $\widetilde{U}(a) = U^G_k$, $w(T_0(a)) = w(T_0)$, and $N_0 = \{a\}$.
For a level i, $0 \leq i \leq k-2$, we compute for each node $s \in N_i$ and each edge $e \in T_0(s)$, the replacement edges $r(e)$ in $T_1(s)$. Node s gives rise to $|T_0(s) \backslash mst(s)|$ children in N_{i+1}. Each such child d, corresponding to an edge e_j in $T_0(s) \backslash mst(s) = \{e_1, \ldots, e_{n-1-|mst(s)|}\}$, is characterized by:

- $mv(d) = mv(s) \cup \{e_j\}$.
- $mst(d) = mst(s) \cup (\cup_{\ell=1}^{j-1} \{e_\ell\})$.
- $\widetilde{U}(d)$ is updated from $\widetilde{U}(s)$ as follows (using Lemma 3):
  - $T_0(d) = T_0(s) \cup \{r(e_j)\} \setminus \{e_j\}$ and hence $w(T_0(d)) = w(T_0(s)) - w(e_j) + w(r(e_j))$.
  - For $j = 1, \ldots, k - |mv(d)|$, $T_j(d)$ is obtained from $T_j(s)$ by deleting the replacement edge e_{rep} of the edge deleted from $T_{j-1}(s)$ and replacing it by its replacement edge $r(e_{rep}) \in T_{j+1}(s)$.
    If for a level i and an edge e_{rep}, the replacement edge $r(e_{rep})$ does not exist, $T_j(d) = T_j(s) \setminus \{e_{rep}\}$ and $T_\ell(d) = T_\ell(s)$ for $\ell = j+1, \ldots, k - |mv(d)|$.

    If for a level i, $T_i(s) = \emptyset$ then $T_\ell(d) = \emptyset$ for $\ell = i, \ldots, k - |mv(d)|$.

At level $k-1$, for each node $s \in N_{k-1}$ and for all edges $e \in T_0(s) \setminus mst(s)$, we find $r(e)$ in $T_1(s)$ and we determine a node s^* that verifies $\max_{s \in N_{k-1}} \max_{e \in T_0(s) \backslash mst(s)} (w(T_0(s)) - w(e) + w(r(e)))$. An optimal solution is the subset $mv(s^*) \cup \{e^*\}$ where $e^* = \arg \max_{e \in T_0(s^*) \backslash mst(s^*)} w(T_0(s^*)) - w(e) + w(r(e))$. The largest weight of a MST in the partial graph obtained by deleting this subset is $w(T_0(s^*)) - w(e^*) + w(r(e^*))$.

Algorithm 1 describes this procedure. Its correctness and complexity are given in Theorem 1.

Theorem 1. *Algorithm 1 computes an optimal solution for an instance of k* Most Vital Edges MST *with n vertices and m edges in $O(km\alpha(m,n) + n^k \log \alpha(2(n-1), n))$ time.*

Algorithm 1. Explicit resolution of k MVE MST

```
   /* Let $a$ be the root of the search tree */
1  Construct $U_k^G$;
2  $mv(a) \leftarrow \emptyset; mst(a) \leftarrow \emptyset; w(T_0(a)) \leftarrow w(T_0); \widetilde{U}(a) \leftarrow U_k^G$;
3  $N_0 \leftarrow \{a\}; N_i \leftarrow \emptyset, i = 1, \ldots, k-1$;
4  for $i \leftarrow 0$ to $k-2$ do
5      forall $s \in N_i$ do
6          forall $e \in T_0(s)$ do
7              find $r(e)$ in $T_1(s)$;
           /* $T_0(s) \backslash mst(s) = \{e_1, \ldots, e_{n-1-|mst(s)|}\}$ */
8          forall $e_j \in T_0(s) \backslash mst(s)$ do
               /* create a new node $d$, a child of $s$ */
9              $mv(d) \leftarrow mv(s) \cup \{e_j\}$;
10             $w(T_0(d)) \leftarrow w(T_0(s)) - w(e_j) + w(r(e_j))$;
11             $mst(d) \leftarrow mst(s) \cup (\cup_{\ell=1}^{j-1}\{e_\ell\})$;
12             determine $\widetilde{U}(d)$;
13             $N_{i+1} \leftarrow N_{i+1} \cup \{d\}$;
14 $max \leftarrow 0$;
15 forall $s \in N_{k-1}$ do
16     forall $e \in T_0(s)$ do
17         find $r(e)$ in $T_1(s)$;
18     forall $e \in T_0(s) \backslash mst(s)$ do
19         if $w(T_0(s)) - w(e) + w(r(e)) > max$ then
20             $max \leftarrow w(T_0(s)) - w(e) + w(r(e))$;
21             $e^* \leftarrow e$;
22             $s^* \leftarrow s$;
   /* The largest weight of a MST in the partial obtained graph is
      $w(T_0(s^*)) - w(e^*) + w(r(e^*))$ */
23 return $S^* = mv(s^*) \cup \{e^*\}$;
```

Proof: We first show that Algorithm 1 gives an optimal solution for k Most Vital Edges MST. Let S^* be the solution returned by Algorithm 1, and w^* the weight of the MST in $U_k^G - S^*$. Consider any solution S', with $|S'| = k$, and w' the weight of the MST in $U_k^G - S'$. Let r be a node of the search tree such that $mv(r) \subseteq S'$ and for any child d of r, $mv(d) \nsubseteq S'$. Clearly, r exists and corresponds at worst to root a when $S' \cap T_0 = \emptyset$. Since, by definition, r is such that no edge of $T_0(r)$ belongs to S', we have $w' = w(T_0(r))$. Moreover, since $w(T_0(r)) \leq w^*$, we have $w' \leq w^*$.

We compute now the complexity of Algorithm 1. The construction of U_k^G requires $O(km\alpha(m,n))$ using k times the best current algorithms for MST [2,13]. Denote by t_u the time for constructing U_k^G, by $t_{edge-rep}$ the time for finding the replacement edges for all edges of a minimum spanning tree, and by t_{gen} the time for generating any node s of the search tree (that is determining $mv(s), mst(s)$ and $\widetilde{U}(s)$). Level 0 requires $|N_0|t_{edge-rep}$ time. Level i takes $|N_i|t_{edge-rep} + |N_i|t_{gen}$ time, for $1 \leq i \leq k-1$. At level k, we compute the k most vital edges. Thus, the total time of Algorithm 1 is given by

$$t_u + \sum_{i=0}^{k-1} |N_i| t_{edge-rep} + \sum_{i=1}^{k-1} |N_i| t_{gen} + |N_k|$$

For each node $s \in N_i$, subset $mv(s)$ consists of ℓ tree edges of $T_0(a)$ and $(i-\ell)$ edges belonging to the union set of the $(i-\ell)$ replacement edges of these ℓ edges, $1 \leq \ell \leq i$ (the p replacement edges of an edge $e \in T_0(a)$ are the p edges of minimum weight which connect the two disconnected components of $T_0(a)\backslash\{e\}$). This implies that $|N_i| = \sum_{\ell=1}^{i} \binom{n-1}{\ell} K_{\ell}^{i-\ell} = \sum_{\ell=1}^{i} \binom{n-1}{\ell}\binom{i-1}{i-\ell} = \binom{n+i-2}{i} = O(n^i)$, where $K_n^p = \binom{n+p-1}{p}$ is the number of combinations with repetition of p elements chosen from a set of n elements.

For a node $s \in N_i$, $1 \leq i \leq k-1$, $\widetilde{U}(s)$ contains at most $k-i+1$ forests. Then, t_{gen} is in $O((k-i+1)n)$ time. Since the replacement edges of a MST in a graph with n vertices and m edges can be computed in $O(m \log \alpha(m,n))$ [12], $t_{edge-rep}$ is in $O(n \log \alpha(2(n-1),n))$ time. Therefore, the complexity of Algorithm 1 is in $O(km\alpha(m,n) + n^k \log \alpha(2(n-1),n))$ time. Note that the time needed to generate all the nodes of the search tree is dominated by the total time to find, for all nodes s of the search tree, the replacement edges $r(e)$ in $T_1(s)$ for all edges $e \in T_0(s)$. □

Remark 1. For each node s of the search tree, we could use, instead of the graph $\widetilde{U}(s)$, the graph $U(s) = U_{k-|mv(s)|}^{G''(s)}$ where $G''(s)$ is the graph obtained from G by contracting the edges of $mst(s)$ and removing the edges of $mv(s)$. Thus, $U(s) = (V, \cup_{i=0}^{k-|mv(s)|} T_i(s))$ where $T_i(s)$ is the MSF of $G''(s) - \cup_{j=0}^{i-1} T_j(s)$. Unfortunately, given a child d of a node s of the search tree, updating efficiently $U(d)$ from $U(s)$ is not as straightforward as for $\tilde{U}$. However, even if updating U could be performed more efficiently than $\tilde{U}$, we would get the same complexity since the time for generating all nodes of the search tree is dominated by the total time for finding the replacement edges for all nodes in the search tree.

Discussion. For fixed k, by using the result of Dixon *et al.* [3], Liang [8] proposes an algorithm to solve k Most Vital Edges MST in $O(n^k \alpha((k+1)(n-1),n))$ time. Using Pettie's result [12] Liang's algorithm can be implemented in $O(t_u + n^k \log \alpha((k+1)(n-1),n))$ time, where t_u is the time for constructing U_k^G. Our algorithm has a complexity that is theoretically slightly worse than that of Liang. Nevertheless, since $\alpha(m,n)$ is always less than or equal to 4 in practice, the complexity of these two algorithms can be considered as equivalent. Moreover, the advantage of our algorithm is to determine, with the same time complexity, an optimal solution for i Most Vital Edges MST, for $1 \leq i \leq k$. Indeed, at each level i, we can find among nodes of N_i, the node with the largest weight of a MST.

For general k, our bound is clearly better than that of Liang. Indeed, in Liang's algorithm, after the determination of U_k^G, Liang divides the problem into two cases: (i) $|T_0 \cap S^*| = i, 1 \leq i < k$ and (ii) $|T_0 \cap S^*| = k$ where S^* represents a subset of k most vital edges. In (i), for every possible combination of i edges among the $n-1$ edges of T_0, $1 \leq i < k$, the author constructs a specific graph $\mathcal{G}$ with a number of nodes and edges depending only on k, and determines the $k-i$ remaining edges in $\mathcal{G}$. In (ii), from every possible choice of $(k-1)$ edges among the $n-1$ edges of T_0, the author constructs a MST T' in the graph obtained by deleting these $(k-1)$ edges and finds the k^{th}

edge to be removed by using the replacement edges of T'. Therefore, (i) and (ii) are performed respectively in $\sum_{i=1}^{k-1}\binom{n-1}{i}(t_{\mathcal{G}}+t_{k-i})$ and $\binom{n-1}{k-1}t_{last}$ time, where $t_{\mathcal{G}}$, t_{k-i} and t_{last} are respectively the time to construct $\mathcal{G}$, the time to determine the $k-i$ remaining edges to be removed from $\mathcal{G}$ and the time to find the k^{th} edge to be removed from $T' \cap T_0$. Note that Liang, who considers only the case where k is fixed, does not need to explicit the term involving t_{k-i}. However, for general k, even if expressing the complexity of his algorithm as in $O(t_u + k^3n^k + \sum_{i=1}^{k-1}\binom{n-1}{i}t_{k-i} + kn^k \log\alpha((k+1)(n-1),n))$, one can observe that it is relatively larger than the complexity of our proposed algorithm that remains in $O(t_u + n^k \log\alpha(2(n-1),n))$ time.

The other exact algorithms proposed in the literature [9,15] have a worse complexity than our algorithm both for fixed ad general k.

4 An Implicit Enumeration Algorithm for Finding the *k* Most Vital Edges

An interesting feature of our explicit enumeration algorithm is that, unlike the algorithms previously proposed, it can easily be adapted to design an implicit algorithm based on a branch and bound scheme. To do this, we use for each node s an upper bound $UB(s)$ based on successive replacements of edges. We also use lower bounds $LB(s)$ constructed by extending the forest, corresponding to s, to a particular minimum spanning tree.

In order to obtain the best possible bounds, we construct $U(s)$ for each node s, instead of using $\tilde{U}(s)$. For each child d of s, $U(d)$ is determined by constructing $T_i(d)$, for $0 \leq i \leq k - |mv(d)|$ from the edges of $U(s)$.

4.1 Lower Bounds

For a fixed node s of the search tree, $k-|mv(s)|$ edges remain to be deleted from $U(s)$. We present different ways of determining these remaining edges giving rise to three possible lower bounds.

1. $LB_{greedy}(s)$: Given $T_0(s)$, we compute $r(e_j)$ for all $e_j \in T_0(s)$. We delete the edge e_j^* which realizes $max_{e_j \in T_0(s)\setminus mst(s)}(w(r(e_j))-w(e_j))$ and replace it by $r(e_j^*)$. We update $U(s)$ and repeat the process until we remove $k-|mv(s)|$ edges. The value of this bound is the weight of the last MST obtained.
2. $LB_{first}(s)$: We remove the $k-|mv(s)|$ edges of $T_0(s) \setminus mst(s)$ having the smallest weight, and we construct a MST from the remaining edges in $T_0(s)$. The weight of the MST obtained is the value of this bound.
3. $LB_{best}(s)$: Given $T_0(s)$, we compute $r(e_j)$ for all $e_j \in T_0(s)$. We remove the $k-|mv(s)|$ edges in $T_0(s) \setminus mst(s)$ whose difference between the weight of their replacement edge and their weight is the largest, and we construct a MST from the remaining edges in $T_0(s)$. The value of this bound is the weight of the MST obtained.

In order to test these bounds, we computed, for instances with different values of n and k, these three lower bounds at the root a of the search tree. The instances are generated as explained in section 6. Due to space limitation, we give in Table 1, results for two types of instances. We note that there is no dominance between these three bounds. We also note that LB_{first} is the fastest in terms of running time but gives bad values. LB_{greedy}, which gives the best values in most cases, takes much more time than the other bounds. LB_{best}, which gives similar values as LB_{greedy}, takes only about twice as much time as LB_{first} and about 40 to 100 times less time than LB_{greedy}.

Table 1. Values of the lower and upper bounds at the root of the search tree

n	k	$LB_{greedy}(a)$		$LB_{first}(a)$		$LB_{best}(a)$		$w(T_0)$	$UB(a)$
		value	time(s)	value	time(s)	value	time(s)	in $G \setminus S^*$	
20	9	265	0.873	255	0.016	250	0.047	282	719
		221	0.889	219	0.015	222	0.032	229	711
		178	0.982	179	0.032	180	0.031	211	669
		166	0.842	157	0.000	157	0.016	186	681
		276	0.624	268	0.015	267	0.016	278	726
		246	0.904	243	0.016	240	0.000	279	764
		236	0.764	232	0.031	235	0.047	239	682
		272	0.967	254	0.031	255	0.031	272	712
		205	1.060	193	0.016	203	0.000	207	668
		245	0.748	216	0.000	225	0.016	249	716
100	7	185	5.912	173	0.032	184	0.062	185	253
		192	5.554	186	0.031	192	0.062	199	264
		215	5.850	192	0.031	212	0.047	215	274
		211	5.585	193	0.031	211	0.062	212	278
		201	5.651	186	0.035	201	0.056	201	265
		215	5.446	194	0.035	215	0.052	215	279
		220	5.028	202	0.034	220	0.052	223	279
		218	5.048	201	0.031	218	0.051	220	284
		202	5.772	192	0.031	202	0.047	204	276
		207	5.616	191	0.031	205	0.047	210	274

4.2 Upper Bound

Let s be a given node of the search tree. To compute $UB(s)$, we select the edge in $T_1(s)$ of largest weight and we replace the edge deleted from $T_j(s)$ by the edge with largest weight belonging to $T_{j+1}(s)$, for $j = 1, \ldots, k - |mv(s)| - 1$. We repeat this process $k - |mv(s)| - 1$ times.

Let F be the set of the $k - |mv(s)|$ edges selected from $T_1(s)$ in this process. Then, we must determine the $k - |mv(s)|$ edges to remove. To obtain an upper bound for all feasible solutions obtained from s, we delete the $k-|mv(s)|$ edges of smallest weight among the edges of $F \cup T_0(s) \setminus mst(s)$. Denote by E_{min} the subset of these selected edges removed. Therefore, $UB(s) = w(T_0(s)) + w(F) - w(E_{min})$.

We computed, for instances with different values of n and k, this upper bound at the root a of the search tree (see Table 1). The main observation is that $UB(a)$ is rather close to the optimal value for small values of k and deteriorates as k increases.

4.3 Branching Strategy

Let a be the root of the search tree. The branching strategy is the same as for the explicit enumeration algorithm. We start with a feasible solution value corresponding to $\max\{LB_{greedy}(a), LB_{first}(a), LB_{best}(a)\}$. We tested two different best first search strategies. The first one is the standard strategy (Branching: best upper bound) where the node with the largest upper bound is selected first. No lower bound is computed and the fathoming test is performed only when we update the current best feasible solution value, which can occur only at level $k-1$ of the search tree. In the second strategy (Branching: best lower bound), the node with the largest lower bound is selected first. Lower and upper bounds are computed at every node. Since LB_{best} gives values close to the best ones and takes less time, we use this bound for computing a lower bound. Here, the fathoming test is performed at each node by comparing each lower bound value with the current best feasible solution value.

5 A Mixed Integer Programming Formulation for Finding the *k* Most Vital Edges

Consider the graph $U_k^G = (V, E_u)$ with $E_u = \cup_{j=0}^{k} T_j$. Let $D = (V, A_u)$ be the digraph obtained by replacing each edge (i,j) in E_u by two arcs (i,j) and (j,i) in A_u and let $w_{ij} = w(e)$ for each edge $e \in E_u$. In [10], Magnanti and Wolsey present a formulation of the minimum spanning tree problem, called the directed multicommodity flow model. Using this model, we propose the following formulation for k Most Vital Edges MST:

$$\begin{cases} \max\limits_{z \in Z} \min \sum\limits_{(i,j)\in E_u} (w_{ij} + M_{ij}\, z_{ij})(y_{ij} + y_{ji}) & \\ \quad \sum\limits_{(j,1)\in A_u} f_{j1}^{\ell} - \sum\limits_{(1,j)\in A_u} f_{1j}^{\ell} = -1 & \forall \ell \in V\backslash\{1\} \\ \quad \sum\limits_{(j,i)\in A_u} f_{ji}^{\ell} - \sum\limits_{(i,j)\in A_u} f_{ij}^{\ell} = 0 & \forall i, \ell \in V\backslash\{1\},\ i \neq \ell \\ \quad \sum\limits_{(j,\ell)\in A_u} f_{j\ell}^{\ell} - \sum\limits_{(\ell,j)\in A_u} f_{\ell j}^{\ell} = 1 & \forall \ell \in V\backslash\{1\} \\ \quad f_{ij}^{\ell} \leq y_{ij} & \forall (i,j) \in A_u,\ \forall \ell \in V\backslash\{1\} \\ \quad \sum\limits_{(i,j)\in A_u} y_{ij} = n-1 & \\ \quad f_{ij} \geq 0, y_{ij} \geq 0 & \forall (i,j) \in A_u \\ \text{where } Z = \{z_{ij} \in \{0,1\},\ \forall (i,j) \in E_u : \sum\limits_{(i,j)\in E_u} z_{ij} = k\} & \end{cases}$$

In this formulation, we consider node 1 as the root of a MST and every node $\ell \neq 1$ defines a commodity. Denote by f_{ij}^{ℓ} the flow of ℓ passing through (i,j). Variable z_{ij} is equal to 1 if edge (i,j) is deleted and 0 otherwise. In order to discard this edge from any MST, we assign it the weight $w_{ij} + M_{ij}$ where M_{ij} is a large enough constant, e.g. $M_{ij} = \max_{(i,j)\in E} w_{ij} + 1 - w_{ij}$.

Using the dual of the inner program, we obtain the following mixed integer programming formulation for k Most Vital Edges MST.

$$\begin{cases}
\max \sum\limits_{\ell \in V,\, \ell \neq 1} (\alpha_\ell^\ell - \alpha_1^\ell) + (n-1)\mu & \\
\sigma_{ij}^\ell \geq \alpha_j^\ell - \alpha_i^\ell & \forall (i,j) \in A_u,\ \forall \ell \in V\backslash\{1\} \\
\sum\limits_{\ell \neq 1} \sigma_{ij}^\ell + \mu \leq w_{ij} + M_{ij}\ z_{ij} & \forall (i,j) \in E_u \\
\sum\limits_{\ell \neq 1} \sigma_{ji}^\ell + \mu \leq w_{ij} + M_{ij}\ z_{ij} & \forall (i,j) \in E_u \\
\sum\limits_{(i,j) \in E_u} z_{ij} = k & \\
z_{ij} \in \{0,1\} & \forall (i,j) \in E_u \\
\sigma_{ij}^\ell \geq 0 & \forall (i,j) \in A_u,\ \forall \ell \in V\backslash\{1\} \\
\alpha_i^\ell \geq 0 & \forall i \in V, \ell \in V\backslash\{1\} \\
\mu \text{ unrestricted} &
\end{cases}$$

6 Computational Results

All experiments presented here were performed on a 3.4GHz computer with 3Gb RAM. All proposed algorithms are implemented in C. All instances are complete graphs defined on n vertices. Weights $w(e)$ for all $e \in E$ are generated randomly, uniformly distributed in $[1, 100]$. For each value of n and k presented in this study, 10 different instances were generated and tested. The results are reported in Table 2 where each given value is the average over 10 instances. For the implicit enumeration algorithm, treated and generated nodes represent respectively nodes for which we have computed mv, mst, and U and nodes satisfying the condition of not fathoming ($UB > bestvalue$). Column ♯opt corresponds to the number of instances solved optimally.

We first compare the explicit and implicit enumeration algorithms. The results show that implicit enumeration algorithms are much faster than the explicit enumeration algorithm and can handle instances of considerably larger size. Observe that, for the explicit enumeration algorithm, the search tree size is identical for any instance of the same (n, k) type. As a consequence, either all or none of the instances of a same (n, k) type can be solved. Moreover, for the same reason, computation times show a low variance for all instances of a same (n, k) type. Regarding the implicit enumeration algorithm, the "Branching: best upper bound" strategy yields slightly better running times than the "Branching: best lower bound" strategy. However, the "Branching: best upper bound" strategy, for which fathoming tests are performed less frequently, generates more nodes. Thus, owing to the limited memory capacity, the "Branching: best lower bound" strategy can handle instances of larger size.

We compare now the results obtained by the mixed integer program with those of the implicit enumeration algorithm. For this, we implemented the mixed integer program using the solver CPLEX 12.1 and we run it on the same generated instances. We limited the running time to 1 hour for the instances with 20, 25, 30 and 50 vertices, and to 2 hours for the other instances. The results are also reported in Table 2 where

- *Time*, given in seconds, is the average running time on the 10 instances. For any instance which is not solved optimally within the time limit, the running time is set to this limit;

- *Generated nodes* represents the average number of nodes created in the search tree corresponding to instances giving a feasible solutions;
- *Gap*, expressed as a percentage, represents the average over ratios $\frac{UB-BS}{UB}$ computed on all instances returning at least one feasible solution, where UB is the final best upper bound and BS is the best solution value found;
- *Opt/Feas* represents the number of instances solved optimally /for which at least one feasible solution was found within the time limit.

We note that the mixed integer program reaches the optimal value for very small instances only. Actually, for $n < 100$, we only obtain in most cases feasible solutions with rather large gaps which indicates that optimality is far from being reached. Finally, for instances with $n \geq 100$, no feasible solutions are returned within the time limit. Moreover, for $n = 300$ and 400, the execution of the program exceeds the memory after a few seconds (297.437 and 0.56 seconds in average respectively).

From all these remarks, we can conclude that our proposed implicit enumeration algorithm gives better results than the explicit enumeration algorithm as well as the resolution of the mixed integer program and this both in terms of running time and using memory capacity.

7 ε-Approximate Algorithm

The proposed algorithm is based on the previous implicit algorithm. The aim being to obtain an ε-approximate solution of the optimum, the condition to generate a node s in the search tree is now $(1-\varepsilon)UB(s) > bestvalue$. Indeed, the value v returned by the approximate algorithm must verify $opt(G)(1-\varepsilon) \leq v \leq opt(G)$. Since v is equal to $bestvalue$, any node for which $UB(s)(1-\varepsilon) \leq bestvalue$ is fathomed.

The algorithm is implemented in C and tested on the same instances generated in Section 6 and this for $\varepsilon = 0.01; 0.05$ and 0.1. Thus, we compare the ε-approximate algorithm with the implicit algorithm. The results are summarized in Table 3. The meaning of treated and generated nodes is the same as in Section 6 and each given value in the table represents the average over the 10 generated instances for each value of n and k.

We note that the running times of the ε-approximate algorithm are significantly lower than those of the implicit enumeration algorithm. Running times do not exceed 21 seconds for $\varepsilon = 0.1$, 180 seconds for $\varepsilon = 0.05$ and 1 215 seconds for $\varepsilon = 0.01$. We also note that for large instances with $n = 300$ and 400 nodes, the ε-approximate algorithm solves the problem for $\varepsilon = 0.05$ and 0.1 at the root in a time less than 1 second, and for $\varepsilon = 0.1$ in a time less than 90 seconds while the implicit enumeration algorithm requires 1 793.460 and 7 265.850 seconds respectively.

Table 2. Comparison of explicit enumeration, implicit enumeration and MIP-based algorithms

n	k	Explicit enumeration		Implicit enumeration							Mixed Integer Program			
				Branching: best lower bound			Branching: best upper bound							
		Time (s)	Nodes	Time (s)	Nodes		Time (s)	Nodes		♯opt	Time (s)	Generated nodes	Gap (%)	Opt/Feas
					Treated	Generated		Treated	Generated					
20	3	**0.000**	210	**0.000**	165.1	33.5	0.001	165.1	34.3	10	35.750	1 638.2	0	10 / 10
	5	0.135	8 855	**0.032**	3 280.6	422.3	**0.032**	3 230.9	463.2	10	692.984	21 792.4	0	10 / 10
	7	2.732	177 100	0.419	35 714.0	4 792.0	**0.380**	35 659.2	5 918.2	10	3 600.000	61 386.5	23.91	0 / 10
	9	36.020	220 075	3.322	258 321.8	35 639.1	**3.047**	257 776.0	44 037.4	10	3 600.000	36 908.1	46.49	0 / 10
25	3	**0.000**	325	**0.000**	245.0	29.8	0.003	245.0	31.4	10	141.270	2 066.5	0	10 / 10
	5	0.318	20 475	0.095	7 146.4	705.1	**0.089**	7 047.2	866.7	10	2 984.021	29 300.4	8.69	5 / 10
	7	8.783	593 775	1.772	128 802.5	15 143.4	**1.617**	128 742.2	16 926.0	10	3 600.000	14 218.1	46.05	0 / 10
	8	52.068	2 629 575	3.765	247 900.6	26 076.8	**3.566**	247 822.8	31 938.3	10	3 600.000	10 733.5	66.43	0 / 10
30	3	0.007	465	**0.000**	345.1	47.7	0.005	345.1	49.7	10	424.171	3 831.9	0	10 / 10
	5	0.812	40 920	0.260	16 756.3	1 373.7	**0.231**	16 625.9	1 588.7	10	3 458.330	13 156.2	26.03	1 / 10
	7	40.461	1 623 160	3.899	231 523.5	20 779.0	**3.553**	231 210.2	25 737.4	10	3 600.000	4 855.9	63.65	0 / 10
50	3	0.880	1 275	0.028	949.1	64.9	**0.026**	949.1	85.3	10	3 600.000	1 285.8	17.28	0 / 10
	5	15.390	292 825	2.043	76 840.3	4 649.3	**1.856**	74 550.2	5 138.1	10	3 600.000	503.0	43.59	0 / 10
	7	-	-	88.886	3 156 471.8	168 127.4	**81.707**	3 156 170.1	218 830.4	10	3 600.000	21.33	80.47	0 / 9
75	3	0.376	2 850	0.101	2 296.8	114.8	**0.096**	2 296.8	117.7	10	7 200.000	430.2	17.83	0 / 10
	5	-	-	11.248	259 738.0	8 130.7	**10.459**	259 737.6	10 519.6	10	6 490.238	0.3	39.22	1 / 10
	7	-	-	**650.008**	13 330 591.9	474 912.7	*463.385*	*9 608 531.7*	*379 179.2*	7	7 200.000	0	55.75	0 / 3
100	3	1.083	5 050	0.224	3 617.1	83.3	**0.210**	3 617.1	89.9	10	7 200.000	0		0 / 0
	5	-	-	54.148	904 662.4	19 383.8	**49.895**	904 662.4	23 800.1	10	7 200.000	0		0 / 0
	7	-	-	**2 016.410**	26 835 600.6	721 120.4	*935.777*	*11 986 049.2*	*368 180.0*	4	7 200.000	0		0 / 0
200	5	-	-	**572.557**	2 933 547.2	46 236.3	670.340	2 933 296.1	49 073.6	10	7 200.000	0		0 / 0
300	5	-	-	**1 793.460**	3 996 192.1	43 671.2	2 163.350	3 980 311.0	56 924.5	10	7 200.000	0		0 / 0
400	5	-	-	**7 265.850**	10 956 321.8	106 433.4	*6 195.182*	*5 927 376.8*	*56 424.5*	7	-	-	-	0 / 0

italics: average over instances solved optimally

-: memory overflow

Table 3. Results of the ε-approximate algorithm

n	k	ε-approximate algorithm											
		$\varepsilon = 0.01$				$\varepsilon = 0.05$				$\varepsilon = 0.1$			
		Time	Nodes		ε'	Time	Nodes		ε'	Time	Nodes		ε'
		(s)	Treated	Generated		(s)	Treated	Generated		(s)	Treated	Generated	
20	3	0.000	162.9	30.6	0.00000	0.000	136.9	14.0	0.00000	0.000	100.6	7.4	0.00198
	5	0.035	3 108.2	384.6	0.00000	0.024	2 068.8	211.1	0.00043	0.012	1 258.4	113.1	0.00267
	7	0.393	33 258.5	4 356.0	0.00000	0.273	21 820.0	2 575.7	0.00323	0.174	13 209.9	1 451.0	0.00922
	9	3.044	237 267.0	32 085.4	0.00000	2.180	160 036.0	20 093.2	0.00421	1.376	93 275.6	10 888.2	0.00735
25	3	0.000	230.8	26.7	0.00000	0.000	189.8	13.1	0.00263	0.000	98.2	5.7	0.00263
	5	0.093	6 691.6	637.2	0.00060	0.061	4 235.5	345.2	0.00213	0.031	2 002.0	146.1	0.00779
	7	1.648	119 033.8	13 603.0	0.00000	1.066	72 193.8	7 178.9	0.00148	0.606	37 683.2	3 379.8	0.00416
	8	3.513	226 536.1	23 389.9	0.00000	2.255	135 623.2	12 792.9	0.00142	1.242	68 426.4	5 900.1	0.00319
30	3	0.000	338.1	38.8	0.00000	0.000	280.1	17.3	0.00453	0.000	161.6	7.2	0.00452
	5	0.233	15 137.4	1 171.7	0.00000	0.123	7 302.6	470.5	0.00307	0.059	3 146.2	181.9	0.00363
	7	3.523	209 289.3	18 256.8	0.00000	2.183	119 797.9	9 363.5	0.00470	1.155	57 665.1	4 062.5	0.00721
50	3	0.025	899.4	48.8	0.00000	0.011	381.6	14.1	0.00000	0.000	76.5	2.4	0.00646
	5	1.790	67 052.0	3 757.3	0.00000	0.635	20 586.6	866.5	0.00178	0.255	7 213.3	241.8	0.00279
	7	74.688	2 534 780.6	130 685.3	0.00000	28.324	820 954.5	36 722.1	0.00053	7.958	193 201.5	7 827.2	0.00316
75	3	0.092	2 121.1	75.7	0.00000	0.0016	325.4	5.3	0.00241	0.003	75.0	1.0	0.00355
	5	8.334	187 230.6	5 444.6	0.00000	1.679	27 753.6	616.2	0.00168	0.232	2 860.8	50.4	0.00387
	7	510.768	9 838 080.8	336 993.8	0.00000	109.664	1 734 007.8	51 514.5	0.00195	20.661	260 410.7	6 584.0	0.00536
100	3	0.208	3 341.4	57.4	0.00000	0.013	121.6	1.4	0.00051	0.010	100.0	1.0	0.00308
	5	34.779	561 343.8	10 619.5	0.00000	3.875	41 860.1	611.1	0.00143	0.396	3 307.4	41.6	0.00297
	7	1 214.43	14 901 505.8	377 861.2	0.00000	179.771	1 703 572.1	34 196.8	0.00143	13.940	95 045.1	1 492.2	0.00371
200	5	165.904	682 703.2	10 147.9	0.00000	0.731	1 693.0	11.5	0.00163	0.131	200.0	1.0	0.00163
300	5	87.600	164 368.6	1 129.4	0.00030	0.380	300.0	1.0	0.00245	0.379	300.0	1.0	0.00241
400	5	89.564	80 786.1	257.3	0.00000	0.846	400.0	1.0	0.00000	0.842	400.0	1.0	0.00000

Moreover, the approximate solutions a posteriori are within ε' to the optimum, with $\varepsilon' \leq 0.0006$ for $\varepsilon = 0.01$, $\varepsilon' \leq 0.0047$ for $\varepsilon = 0.05$ and $\varepsilon' \leq 0.00922$ for $\varepsilon = 0.1$.

For $\varepsilon = 0.01$, we note that the problem is nearly solved to optimality ($\varepsilon' = 0$).

All these remarks show that the proposed lower bounds and upper bound are of very good quality and that the running time of the implicit enumeration algorithm is the time needed to verify the optimality of the solution. Indeed, this optimal solution is either found in a few seconds or determined at the root of the search tree corresponding then to the maximum value of the three lower bounds associated to the root.

8 Conclusions

Algorithms proposed in this paper can be easily adapted to solve some variants of the k Most Vital Edges MST problem. In a first variant, a removing cost is associated to each edge. The problem consists of finding a subset of edges with total cost bounded by a budget limit whose deletion causes the largest increase in the weight of a minimum spanning tree. In a second variant, we have to determine a minimum number of edges to be removed such that the weight of a minimum spanning tree in the resulting graph is at least a fixed value.

References

1. Bar-Noy, A., Khuller, S., Schieber, B.: The complexity of finding most vital arcs and nodes. Technical Report CS-TR-3539, University of Maryland (1995)
2. Chazelle, B.: A minimum spanning tree algorithm with inverse-Ackermann type complexity. Journal of the ACM 47(6), 1028–1047 (2000)
3. Dixon, B., Rauch, M., Tarjan, R.E.: Verification and sensitivity analysis of minimum spanning trees in linear time. SIAM Journal on Computing 21(6), 1184–1192 (1992)
4. Frederickson, G.N., Solis-Oba, R.: Increasing the weight of minimum spanning trees. In: Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms (SODA 1996), pp. 539–546 (1996)
5. Hsu, L., Jan, R., Lee, Y., Hung, C., Chern, M.: Finding the most vital edge with respect to minimum spanning tree in a weighted graph. Information Processing Letters 39(5), 277–281 (1991)
6. Iwano, K., Katoh, N.: Efficient algorithms for finding the most vital edge of a minimum spanning tree. Information Processing Letters 48(5), 211–213 (1993)
7. Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V., Rudolf, G., Zhao, J.: On short paths interdiction problems: total and node-wise limited interdiction. Theory of Computing Systems 43(2), 204–233 (2008)
8. Liang, W.: Finding the k most vital edges with respect to minimum spanning trees for fixed k. Discrete Applied Mathematics 113(2-3), 319–327 (2001)
9. Liang, W., Shen, X.: Finding the k most vital edges in the minimum spanning tree problem. Parallel Computer 23(3), 1889–1907 (1997)

10. Magnanti, T.L., Wolsey, L.: Optimal trees. In: Ball, M.O., et al. (eds.) Network Models. Handbook in Operations Research and Management Science, vol. 7, pp. 503–615. North-Holland, Amsterdam (1995)
11. Nardelli, E., Proietti, G., Widmayer, P.: A faster computation of the most vital edge of a shortest path. Information Processing Letters 79(2), 81–85 (2001)
12. Pettie, S.: Sensitivity analysis of minimum spanning tree in sub-inverse-ackermann time. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 964–973. Springer, Heidelberg (2005)
13. Pettie, S., Ramachandran, V.: An optimal minimum spanning tree algorithm. Journal of the ACM 49(1), 16–34 (2002)
14. Ratliff, H.D., Sicilia, G.T., Lubore, S.H.: Finding the n most vital links in flow networks. Management Science 21(5), 531–539 (1975)
15. Shen, H.: Finding the k most vital edges with respect to minimum spanning tree. Acta Informatica 36(5), 405–424 (1999)
16. Suraweera, F., Maheshwari, P., Bhattacharya, P.: Optimal algorithms to find the most vital edge of a minimum spanning tree. Technical Report CIT-95-21, School of Computing and Information Technology, Griffith University (1995)
17. Tarjan, R.E.: Applications of path compression on balanced trees. Journal of the ACM 26(4), 690–715 (1979)
18. Wollmer, R.: Removing arcs from a network. Operations Research 12(6), 934–940 (1964)
19. Wood, R.K.: Deterministic network interdiction. Mathematical and Computer Modeling 17(2), 1–18 (1993)

Euclidean Chains and Their Shortcuts

Boting Yang

Department of Computer Science, University of Regina
boting@cs.uregina.ca

Abstract. A Euclidean graph is a straight line embedding of a graph in the plane such that there is no crossing between any pair of edges and the length of an edge is the Euclidean distance between its two endpoints. A Euclidean chain $C = (x_1, x_2, \ldots, x_n)$ is a planar straight line graph with vertex set $\{x_1, x_2, \ldots, x_n\}$ and edge set $\{x_i x_{i+1} : 1 \leq i \leq n-1\}$. Given a Euclidean chain C in the plane, we study the problem of finding a pair of points on C such that the new Euclidean graph obtained from C by adding a straight line segment (called shortcut) between this pair of points has the minimum diameter. We also study the ratio between the diameter of the new graph and the length of C. We give necessary and sufficient conditions for optimal shortcuts. We present three approximation algorithms for computing the optimal shortcuts of chains. One of them is a fully polynomial-time approximation scheme (FPTAS). We introduce two types of chains, strongly monotonic chain and simple chain. We provide properties for these two types of chains and their shortcuts.

1 Introduction

A Euclidean graph is a straight line embedding of a graph in the plane such that there is no crossing between any pair of edges and the length of an edge is the Euclidean distance between its two endpoints. Diameter is an important parameter for Euclidean graphs. For a Euclidean graph G with vertex set $V(G)$ and edge set $E(G)$, we also use G to denote the set of all points on the Euclidean graph G. Thus, $V(G) \subseteq G$. When we use G as a point set, we treat it as a closed point set. A *path* between two points u and v in a Euclidean graph G is a sequence $ux_1x_2 \ldots x_kv$ such that x_i are distinct vertices in $V(G)$, $x_ix_{i+1} \in E(G)$, $1 \leq i \leq k-1$, u is a point on an edge ($\neq x_1x_2$) incident with x_1, and v is a point on an edge ($\neq x_{k-1}x_k$) incident with x_k. The distance between u and v is the length of the shortest path between u and v in G, denoted as $\text{dist}_G(u, v)$.

The *diameter* of a Euclidean graph G, denoted as $\text{diam}(G)$, is defined as $\max\{\text{dist}_G(u, v)$: u and v are two points in $G\}$. Note that the diameter of G always exists because G is a closed set when it is considered as a point set in the plane. A *diameter path* of G is a path in G such that the length of this path is $\text{diam}(G)$. Leizhen Cai [3] proposed the following minimum diameter problem: Given a Euclidean graph G, find two points x and y in G such that the new graph obtained from G by adding the segment xy has the minimum diameter.

A *Euclidean chain* $(x_1, x_2, \ldots, x_n)$ is a planar straight line graph with n distinct vertices x_i $(1 \leq i \leq n)$ and $n-1$ edges $x_i x_{i+1}$ $(1 \leq i \leq n-1)$. Euclidean chains are also called simple polygonal chains, polylines, or linkages in the literature. We will simply call them chains in the remainder of the paper. Note that there is no crossing between any pair of edges of a chain.

Given a chain C in the plane, we study the problem of finding a pair of points on C such that the new Euclidean graph obtained from C by adding a straight line segment between this pair of points has the minimum diameter. This problem is relevant to the geometric minimum-diameter spanning tree problem. Given a set P of points in the plane, a geometric minimum-diameter spanning tree of P is a spanning tree of P such that the longest path through the tree is minimized. Ho et al. [5] describe an algorithm that uses $O(n^3)$ time to generate a geometric minimum-diameter spanning tree of n points. Spriggs et al. [7] give an algorithm that generates a tree whose diameter is no more than $(1+\epsilon)$ times that of a geometric minimum-diameter spanning tree, for any $\epsilon > 0$. Gudmundsson et al. [4] show that this problem can be solved in $O(n^2 \log n)$ time and that it yields a factor-4/3 approximation of a geometric minimum-diameter spanning tree.

2 Preliminaries

Throughout this paper, let C be a chain in the plane with at least three distinct vertices satisfying that no three vertices are collinear and no two edges are parallel. Note that we use this assumption only to simplify cases in proofs. We can extend all results to general chains without technical difficulties. For any two points $u, v \in C$, the subchain of C between u and v is denoted by $\text{chain}_C(u, v)$. In the case with no ambiguity, we use $\text{chain}(u, v)$ without subscripts. For convenience, u is referred to the *left end* and v is referred to the *right end* of $\text{chain}(u, v)$. So, a chain $(x_1, x_2, \ldots, x_n)$ is also denoted by $\text{chain}(x_1, x_n)$, where x_1 is the left end and x_n is the right end. A chain C is *monotonic* if there exists a straight line L such that a line orthogonal to L intersects C in at most one point. We also say that C is monotonic w.r.t. L. Monotonic chains have been widely used in the literature of computational geometry[6]. We introduce two variants of monotonic chains, one is a special case, the other is an extension. A chain C is called *strongly monotonic* if for any edge pq of C and any point $v \in pq$, the orthogonal line to pq and passing through v intersects C only at the point v. A chain C is said to be *simple* if for any point $u \in C$, there is a straight line L such that $C \cap L = \{u\}$. It is easy to see that all monotonic chains are also simple chains, but the reverse is not true.

For two points u and v in the plane, the straight line passing through u and v is denoted as $\langle uv \rangle$, and the straight line segment between u and v is denoted as uv. We also use uv to denote the set of all points on the segment uv. We use $\text{int}(uv)$ to denote the set of all interior points on the segment uv. The length of a segment uv is denoted by $|uv|$. An edge between two vertices $u, v \in V(G)$ is the straight line segment uv with two endpoints u and v.

For a chain C and two points $u, v \in C$, the segment uv is called a *shortcut* of C if u and v are isolated points in the intersection point set $uv \cap C$. Furthermore, if $|uv \cap V(C)| = k$, then we call uv a *node-k shortcut*. The number of intersection points between C and uv is called the *size* of the shortcut. We call uv a *maximal shortcut* of C if there does not exist a shortcut S of C such that S contains uv. We call uv a *simple shortcut* if $uv \cap C = \{u, v\}$. For a simple chain with left end a and right end b and a shortcut uv of chain(a, b), if $\text{dist}_C(a, u) < \text{dist}_C(a, v)$, then u is referred to the *left end* and v is referred to the *right end* of the shortcut uv.

Let $C = \text{chain}(a, b)$, and xy be a shortcut of C such that $xy \cap C = \{x_1, x_2, \ldots, x_k\}$, where $\text{dist}_C(a, x_1) < \text{dist}_C(a, x_2) < \cdots < \text{dist}_C(a, x_k)$. We use $C \cup xy$ to denote the Euclidean graph obtained by adding vertices $x_1(= x)$, $x_2, \ldots, x_k(= y)$ and edges $x_1x_2, \ldots, x_{k-1}x_k$ to C. If $xy \cap C$ contains an edge of C, we can define $C \cup xy$ similarly. The shortcut xy partitions C into a sequence of subchains $\text{chain}_C(a, x_1)$, $\text{chain}_C(x_1, x_2)$, $\ldots$, $\text{chain}_C(x_k, b)$.

Given a chain C, the problem of the minimum diameter with shortcut, briefly *MDS(C)*, is to find a shortcut x^*y^* of C such that $\text{diam}(C \cup x^*y^*) \leq \text{diam}(C \cup xy)$ for any xy that is a shortcut of C. Such a shortcut x^*y^* is called an *optimal shortcut* of C or *optimal solution* of MDS(C).

Let S be a straight line or a shortcut of a chain $C(= \text{chain}(a, b))$. The *dual* of the intersection point set $S \cap C$ is the set of edges of C such that each edge contains at least one point in $S \cap C$. For simplicity, the dual of $S \cap C$ is also called the dual of S and is denoted by S_{dual}. Recall that no three vertices of C are collinear. If every point in $S \cap C$ is an interior point of an edge of C and $|S \cap C| = k$, then S_{dual} contains k edges of C. If $S \cap C$ contains only one vertex in $V(C) \setminus \{a, b\}$ and $|S \cap C| = k$, then S_{dual} contains $k + 1$ edges of C. If $S \cap C$ contains two vertices in $V(C) \setminus \{a, b\}$ and $|S \cap C| = k$, then S_{dual} contains $k + 2$ edges of C. If $S \cap C$ contains an edge of C and k interior points of edges of C, then S_{dual} contains $k + 3$ edges of C.

3 Properties

In this section, we give characterizations of strongly monotonic chains and simple chains. We also give some properties of shortcuts.

Theorem 1. *A chain C is strongly monotonic if and only if for any two points $u, v \in C$, C is monotonic w.r.t. the straight line $\langle uv \rangle$.*

Proof. The sufficiency is easy to see. We only show the necessity by contradiction. Let C be a strongly monotonic chain. Suppose there are two points $u, v \in C$ such that C is not monotonic w.r.t. the straight line $\langle uv \rangle$. Then, there is a point w on $\langle uv \rangle$ such that the line L_w passing through w and perpendicular to $\langle uv \rangle$ intersects C in at least two points, say p and q. Note that the subchain $\text{chain}_C(p, q)$ is on a simple polygon formed by $\text{chain}_C(u, v)$ and uv. Since $\langle pq \rangle$ is orthogonal to $\langle uv \rangle$, there is an edge on $\text{chain}_C(p, q)$ such that a vertical line of this edge intersects C in at least two points. This is a contradiction. Thus, for any two points $u, v \in C$, C is monotonic w.r.t. the straight line $\langle uv \rangle$.

Theorem 2. *If C is a strongly monotonic chain, then for any subchain* $\mathrm{chain}_C(u,v)$ *and any point* $p \in \mathrm{chain}_C(u,v)$*,* $\max\{|up|,|vp|\} \le |uv|$*.*

Proof. Let L be a moving line which is always vertical to edges of C when it moves from u to v. When L passes any interior point p on an edge xy of $\mathrm{chain}_C(u,v)$, it must be a bisector of some pair of points x' and y' on xy (x' is on the left side and y' on the right side). Since L intersects C at only one point, we know that $|uy'| \ge |up|$ and $|vx'| \ge |vp|$. When L moves from u to v, by the transitivity of inequalities, we know that $\max\{|up|,|vp|\} \le |uv|$.

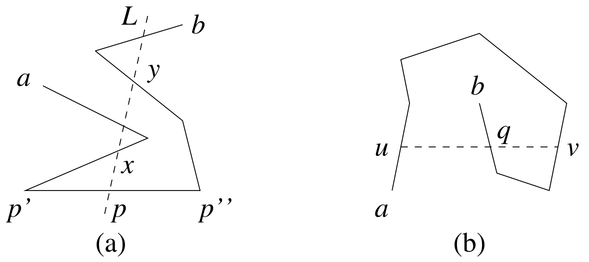


Fig. 1. Theorem 3

Theorem 3. *A chain C is simple if and only if for any two points $u, v \in C$, all intersection points between the straight line segment uv and C lie on the subchain between u and v.*

Proof. (Sufficiency.) Let p be an arbitrary point on C, $p'p''$ be the edge of C containing p and L be a straight line containing p but not containing $p'p''$. Suppose that L intersects C at more than one point. We will show that we can always rotate L around p to find a position such that L intersects C only at the point p. We first show by contradiction that L cannot intersect both subchains $\mathrm{chain}_C(a,p)$ and $\mathrm{chain}_C(p,b)$ in the same open half plane divided by $\langle p'p''\rangle$. Suppose that in the same open half plane, L intersects $\mathrm{chain}_C(a,p)$ at x and intersects $\mathrm{chain}_C(p,b)$ at y (see Figure 1(a)). If $x \in py$, then $x \in C \cap py$. This is a contradiction because $x \notin \mathrm{chain}_C(p,y)$. If $y \in xp$, then $y \in C \cap xp$. This is a contradiction because $y \notin \mathrm{chain}_C(x,p)$. Thus, L cannot intersect both subchains $\mathrm{chain}_C(a,p)$ and $\mathrm{chain}_C(p,b)$ in the same open half plane divided by $\langle p'p''\rangle$. Similarly, we can show that L cannot intersect the subchain $\mathrm{chain}_C(a,p)$ (or $\mathrm{chain}_C(p,b)$) in both open half planes divided by $\langle p'p''\rangle$. Therefore, when we rotate L around p, we can find a position such that L intersects C only at p.

(Necessity.) Suppose that C has two points u and v such that there is a point $q \in uv \cap C$ and $q \notin \mathrm{chain}_C(u,v)$ (see Figure 1(b)). Then for any straight line passing through q must intersect a point in $\mathrm{chain}_C(u,v)$. This is a contradiction.

Lemma 1. *For any two points u and v on a chain C,* $\mathrm{diam}(C \cup uv) \le \mathrm{diam}(C)$*.*

Note that Lemma 1 cannot be extended to graphs, even trees. For example, consider a tree T with edges ab, ac and ad satisfying $|ab| = |ac| = |bc| = 1$, $|ad| = 0.8$ and $ad \cap bc = \emptyset$. Then $\mathrm{diam}(T) = |ab| + |ac| = 2$. But $\mathrm{diam}(T \cup bc) = |ad| + |ab| + |bc|/2 = 2.3$.

Lemma 2. *For any two points u and v in a chain C, if there are two points $u', v' \in C$ such that the segment $u'v'$ contains uv, then* $\mathrm{diam}(C \cup u'v') \leq \mathrm{diam}(C \cup uv)$.

Corollary 1. *For any chain C, there is a maximal shortcut which is an optimal solution of MDS(C).*

Let S be a shortcut of a chain C. Recall that S_{dual} is the set of edges of C such that each edge contains at least one point in $S \cap C$. For maximal shortcuts of C, we can show that the number of different duals is bounded by $2n(n-1)$.

Theorem 4. *For a chain C with n vertices, the set $\{S_{\mathrm{dual}}$: S is a maximal shortcut of $C\}$ has at most $2n(n-1)$ elements.*

Lemma 3. *For any simple chain C with left end a and right end b, it has the following two properties.*

(i) *There exist two unique points $a^*, b^* \in C$ such that* $\mathrm{dist}_C(a, a^*) = \mathrm{dist}_C(b, b^*)$ *and* $\mathrm{dist}_{C \cup a^*b^*}(a, b) = \mathrm{dist}_C(a^*, b^*)$.

(ii) *For any shortcut uv of C with left end u and right end v satisfying that* $\mathrm{dist}_C(a, u) = \mathrm{dist}_C(b, v)$, *if* $\mathrm{dist}_C(a, u) < \mathrm{dist}_C(a, a^*)$, *then* $\mathrm{dist}_{C \cup uv}(a, b) < \mathrm{dist}_C(u, v)$, *and if* $\mathrm{dist}_C(a, u) > \mathrm{dist}_C(a, a^*)$, *then* $\mathrm{dist}_{C \cup uv}(a, b) > \mathrm{dist}_C(u, v)$.

Proof. For any shortcut uv of C with left end u and right end v satisfying that $\mathrm{dist}_C(a, u) = \mathrm{dist}_C(b, v)$, let $f(u, v) = \mathrm{dist}_{C \cup uv}(a, b)$ and $g(u, v) = \mathrm{dist}_C(u, v)$. Since C is a simple chain, we know that $\mathrm{chain}_C(a, u) \cap uv = \{u\}$ and $\mathrm{chain}_C(b, v) \cap uv = \{v\}$. Thus, $\mathrm{dist}_{C \cup uv}(a, b) = \mathrm{dist}_C(a, u) + |uv| + \mathrm{dist}_C(b, v)$. When u and v move with the same speed from the ends a and b, respectively, towards the mid-point of C, the function $f(u, v)$ is strictly increasing from $|ab|$ to $\mathrm{dist}_C(a, b)$, and the function $g(u, v)$ is strictly decreasing from $\mathrm{dist}_C(a, b)$ to 0. Thus, there exist two unique points $a^*, b^* \in C$ such that $\mathrm{dist}_C(a, a^*) = \mathrm{dist}_C(b, b^*)$ and $f(a^*, b^*) = g(a^*, b^*)$; furthermore, $f(u, v) < g(u, v)$ when $\mathrm{dist}_C(a, u) < \mathrm{dist}_C(a, a^*)$, and $f(u, v) > g(u, v)$ when $\mathrm{dist}_C(a, u) > \mathrm{dist}_C(a, a^*)$.

4 Simple Shortcuts

Given a chain C with an optimal shortcut S^*, we call S^* an *optimal simple shortcut* of C if the optimal shortcut S^* is also a simple shortcut. In this section we consider the problem of finding optimal simple shortcuts. Note that some chains may have no optimal simple shortcut.

From Lemma 3, we can see the importance of a^* and b^* regarding the diameter of the new graph obtained by adding a shortcut.

Definition 1. For a simple chain C with left end a and right end b, if C has a shortcut a^*b^* with left end a^* and right end b^* such that $\mathrm{dist}_C(a, a^*) = \mathrm{dist}_C(b, b^*)$ and $\mathrm{dist}_{C \cup a^*b^*}(a, b) = \mathrm{dist}_C(a^*, b^*)$, then the two points a^* and b^* are called the *left critical point* and the *right critical point* of C, respectively.

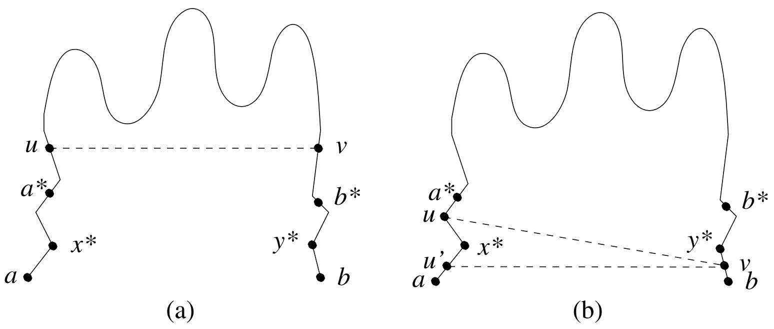


Fig. 2. (a) $v \in \text{chain}_C(a^*, b^*)$. (b) $v \in \text{chain}_C(b^*, b)$.

Theorem 5. *Let* $C = \text{chain}(a, b)$ *be a simple chain with left critical point* a^* *and right critical point* b^* *such that for any two points* $p \in \text{chain}_C(a, a^*)$ *and* $q \in \text{chain}_C(b^*, b)$ *with* $\text{dist}_C(a, p) = \text{dist}_C(q, b)$*,* pq *is a simple shortcut of* C*. Let* $E_c = \{pq : p \in \text{chain}_C(a, a^*),\ q \in \text{chain}_C(b^*, b)$ *and* $\text{dist}_C(a, p) = \text{dist}_C(q, b)\}$*. If* $x^*y^* \in E_c$ *is the segment with the minimum length among all segments in* E_c*, then* $\text{diam}(C \cup x^*y^*) \leq \text{diam}(C \cup uv)$ *for any shortcut* uv *of* C*.*

Proof. Note that all elements in E_c are simple shortcuts of C. From Lemma 3, we have $\text{dist}_{C \cup x^*y^*}(a, b) = 2\text{dist}_C(a, x^*) + |x^*y^*| \leq \text{dist}_C(x^*, y^*)$. Thus, $\text{dist}_C(a, x^*)$ $+ \frac{1}{2}(\text{dist}_C(x^*, y^*) + |x^*y^*|) \geq 2\text{dist}_C(a, x^*) + |x^*y^*|$. Hence,

$$\text{diam}(C \cup x^*y^*) = \text{dist}_C(a, x^*) + \frac{1}{2}(\text{dist}_C(x^*, y^*) + |x^*y^*|).$$

Let uv be an arbitrary shortcut of C with left end u and right end v. We want to prove that $\text{diam}(C \cup uv) \geq \text{diam}(C \cup x^*y^*)$. Without loss of generality, suppose that $\text{dist}_C(a, u) \geq \text{dist}_C(v, b)$. There are two cases regarding the location of v.

Case 1. $v \in \text{chain}_C(a^*, b^*)$ (see Figure 2(a)). Since $\text{dist}_C(a, u) < \text{dist}_C(a, v)$ and $\text{dist}_C(a, u) \geq \text{dist}_C(v, b)$, we know that $u \in \text{chain}_C(a^*, b^*)$. Thus

$$\begin{aligned}
\text{diam}(C \cup uv) &\geq \text{dist}_C(a, u) + |uv| + \text{dist}_C(b, v) \\
&\geq 2\text{dist}_C(a, a^*) + |a^*b^*| \\
&= \text{dist}_C(a, a^*) + \tfrac{1}{2}(\text{dist}_C(a^*, b^*) + |a^*b^*|) \\
&= \text{dist}_C(a, x^*) + \tfrac{1}{2}(\text{dist}_C(x^*, y^*) + |x^*y^*|) \\
&\quad + \tfrac{1}{2}(|a^*b^*| - |x^*y^*|) \\
&\geq \text{dist}_C(a, x^*) + \tfrac{1}{2}(\text{dist}_C(x^*, y^*) + |x^*y^*|) \\
&= \text{diam}(C \cup x^*y^*).
\end{aligned}$$

Case 2. $v \in \text{chain}_C(b^*, b)$ (see Figure 2(b)). Let $u' \in \text{chain}_C(a, a^*)$ such that $\text{dist}_C(a, u') = \text{dist}_C(v, b)$. Since $u'v \in E_c$, we have

$$\begin{aligned}\operatorname{diam}(C \cup uv) &\geq \operatorname{dist}_C(a,u) + \tfrac{1}{2}(\operatorname{dist}_C(u,v) + |uv|) \\ &= \operatorname{dist}_C(a,u') + \tfrac{1}{2}(\operatorname{dist}_C(u',v) + |u'v|) \\ &\quad + \tfrac{1}{2}(\operatorname{dist}_C(u',u) + |uv| - |u'v|) \\ &\geq \operatorname{dist}_C(a,u') + \tfrac{1}{2}(\operatorname{dist}_C(u',v) + |u'v|) \\ &= \operatorname{dist}_C(a,x^*) + \tfrac{1}{2}(\operatorname{dist}_C(x^*,y^*) + |x^*y^*|) \\ &\quad + \tfrac{1}{2}(|u'v| - |x^*y^*|) \\ &\geq \operatorname{dist}_C(a,x^*) + \tfrac{1}{2}(\operatorname{dist}_C(x^*,y^*) + |x^*y^*|) \\ &= \operatorname{diam}(C \cup x^*y^*).\end{aligned}$$

From the above cases, we know that $\operatorname{diam}(C \cup x^*y^*) \leq \operatorname{diam}(C \cup uv)$ for any segment uv with $u, v \in C$.

Note that all chains considered in this paper have at least three distinct vertices satisfying that no three vertices are collinear and no two edges are parallel. Thus, any convex simple chain always has a unique optimal shortcut.

Corollary 2. *Let C be a chain with left end a and right end b such that the polygon formed by connecting a and b is a convex polygon. Let a^* be the left critical point and b^* be the right critical point of C.*

(i) *If $|a^*b^*| < |ab|$, then a^*b^* is a unique optimal shortcut of C.*
(ii) *If $|a^*b^*| \geq |ab|$, then ab is a unique optimal shortcut of C.*

As an application of Corollary 2, let us consider a chain that consists of two edges.

Corollary 3. *Let C be a chain consisting of two edges ac and bc. Then there exist four points $a', a'' \in ac$ and $b', b'' \in bc$ such that $|aa'| = |a'a''| = |bb'| = |b'b''|$ and $|a'b'| = |a''c| + |cb''|$, moreover, $a'b'$ is a unique optimal shortcut of C.*

Note that the optimal solution of the MDS problem may not be unique. For example, if $C = \operatorname{chain}(a, b)$ is a simple chain satisfying the condition in Theorem 5, and uv and xy are two segments in E_c with the minimum length among all segments in E_c, then from Theorem 5, we know that both uv and xy are optimal shortcuts of C. However, the optimal solution is unique for shortcuts with the same dual.

Theorem 6. *For a simple chain C, the optimal shortcut of C is unique among all shortcuts with the same dual.*

5 Node-0 Shortcuts

In Section 4, we give sufficient conditions for optimal simple shortcuts. In this section, we consider necessary conditions of the optimal shortcut S^* of a simple chain C satisfying that $|S^* \cap V(C)| = 0$. We first consider necessary conditions of optimal node-0 shortcuts of size 2. Note that a node-0 shortcut of size 2 is always a simple shortcut, but a simple shortcut may not be a node-0 shortcut of size 2.

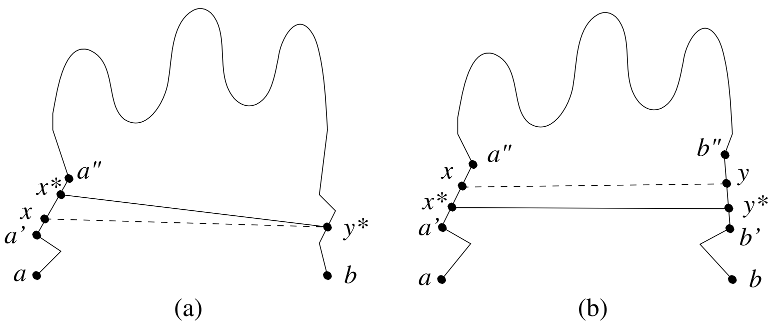


Fig. 3. Node-0 shortcuts of size 2

Theorem 7. *Let C be a simple chain with left end a and right end b. If C has an optimal shortcut x^*y^* with left end x^* and right end y^* such that $|x^*y^* \cap V(C)| = 0$ and $x^*y^* \cap C = \{x^*, y^*\}$, then x^* is the left critical point and y^* is the right critical point of C.*

Proof. From Definition 1, we need to show that $\text{dist}_C(a, x^*) = \text{dist}_C(b, y^*)$ and $\text{dist}_C(x^*, y^*) = \text{dist}_{C\cup x^*y^*}(a, b)$. Let $a'a''$ and $b'b''$ be two edges of C such that $x^* \in \text{int}(a'a'')$, $\text{dist}_C(a, a') < \text{dist}_C(a, a'')$, $y^* \in \text{int}(b'b'')$, and $\text{dist}_C(b, b') < \text{dist}_C(b, b'')$. We first show that $\text{dist}_C(a, x^*) = \text{dist}_C(b, y^*)$ by contradiction. Suppose that $\text{dist}_C(a, x^*) > \text{dist}_C(b, y^*)$ (see Figure 3(a)). Since $x^*y^* \cap C = \{x^*, y^*\}$, there is a point $x \in \text{int}(a'x^*)$ such that $\text{dist}_C(a, x) > \text{dist}_C(b, y^*)$ and $xy^* \cap C = \{x, y^*\}$. Since $|xy^*| < |xx^*| + |x^*y^*|$, we have

$$\text{dist}_C(a, x) + |xy^*| + \text{dist}_C(b, y^*) < \text{dist}_C(a, x^*) + |x^*y^*| + \text{dist}_C(b, y^*),$$

$$\text{dist}_C(a, x) + \frac{1}{2}(\text{dist}_C(x, y^*) + |xy^*|) < \text{dist}_C(a, x^*) + \frac{1}{2}(\text{dist}_C(x^*, y^*) + |x^*y^*|).$$

Thus, $\text{diam}(C\cup xy^*) < \text{diam}(C\cup x^*y^*)$. This is a contradiction. Hence, $\text{dist}_C(a, x^*) \le \text{dist}_C(b, y^*)$. Similarly, we can show that $\text{dist}_C(b, y^*) \le \text{dist}_C(a, x^*)$. Therefore, $\text{dist}_C(a, x^*) = \text{dist}_C(b, y^*)$.

We now show that $\text{dist}_C(x^*, y^*) = \text{dist}_{C\cup x^*y^*}(a, b)$. Suppose $\text{dist}_C(x^*, y^*) > \text{dist}_{C\cup x^*y^*}(a, b)$ (see Figure 3(b)). Then $\text{dist}_C(a, x^*) + \frac{1}{2}(\text{dist}_C(x^*, y^*) + |x^*y^*|) > \text{dist}_{C\cup x^*y^*}(a, b)$. Thus, $\text{diam}(C\cup x^*y^*) = \text{dist}_C(a, x^*) + \frac{1}{2}(\text{dist}_C(x^*, y^*) + |x^*y^*|)$. Since $x^*y^* \cap C = \{x^*, y^*\}$ and the two edges $a'a''$ and $b'b''$ are not parallel, there are two points $x \in \text{int}(a'a'')$ and $y \in \text{int}(b'b'')$ such that $\text{dist}_C(a, x) = \text{dist}_C(b, y)$, $xy \cap C = \{x, y\}$ and $|xy| < |x^*y^*|$. Then we have

$$\begin{aligned}\text{diam}(C \cup x^*y^*) &= \text{dist}_C(a, x^*) + \tfrac{1}{2}(\text{dist}_C(x^*, y^*) + |x^*y^*|)\\ &> \text{dist}_C(a, x) + \tfrac{1}{2}(\text{dist}_C(x, y) + |xy|)\\ &= \text{diam}(C \cup xy).\end{aligned}$$

This is a contradiction. Thus, $\text{dist}_C(x^*, y^*) \le \text{dist}_{C\cup x^*y^*}(a, b)$.

Suppose $\text{dist}_C(x^*, y^*) < \text{dist}_{C\cup x^*y^*}(a, b)$. Then $\text{dist}_C(a, x^*) + \frac{1}{2}(\text{dist}_C(x^*, y^*) + |x^*y^*|) < \text{dist}_{C\cup x^*y^*}(a, b)$. Thus, $\text{diam}(C\cup x^*y^*) = \text{dist}_{C\cup x^*y^*}(a, b)$. Since $x^*y^* \cap$

$C = \{x^*, y^*\}$, there are two points $u \in \text{int}(a'x^*)$, $v \in \text{int}(b'y^*)$ such that $\text{dist}_C(a,u) = \text{dist}_C(b,v)$, $uv \cap C = \{u,v\}$ and $\text{dist}_C(u,v) < \text{dist}_{C\cup uv}(a,b)$. Then we have $\text{diam}(C \cup x^*y^*) = \text{dist}_{C\cup x^*y^*}(a,b) > \text{dist}_{C\cup uv}(a,b) = \text{diam}(C \cup uv)$. This is a contradiction. Hence, $\text{dist}_C(x^*,y^*) = \text{dist}_{C\cup x^*y^*}(a,b)$.

Definition 2. Let C be a simple chain with left end a and right end b, and $v \in C$. A point $a_v^* \in \text{chain}(a,v)$ is called the *left pivot* of v if $2\text{dist}_C(a,a_v^*) + |a_v^*v| = \text{dist}_C(a_v^*,v)$. A point $b_v^* \in \text{chain}(v,b)$ is called the *right pivot* of v if $2\text{dist}_C(b,b_v^*) + |b_v^*v| = \text{dist}_C(b_v^*,v)$.

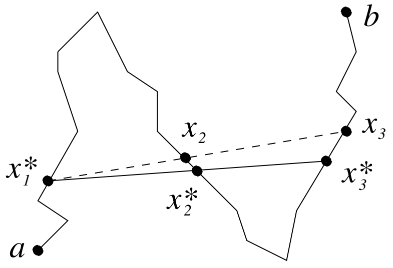


Fig. 4. Node-0 shortcuts of size 3

We now consider necessary conditions of optimal node-0 shortcuts of size 3.

Theorem 8. *Let C be a simple chain with left end a and right end b. If C has an optimal shortcut $x_1^*x_3^*$ with left end x_1^* and right end x_3^* such that $|x_1^*x_3^* \cap V(C)| = 0$ and $x_1^*x_3^* \cap C = \{x_1^*, x_2^*, x_3^*\}$, then x_1^* is the left pivot of x_2^* and x_3^* is the right pivot of x_2^*.*

Proof. From Definition 2, we need to show that $2\text{dist}_C(a,x_1^*) + |x_1^*x_2^*| = \text{dist}_C(x_1^*,x_2^*)$ and $2\text{dist}_C(b,x_3^*) + |x_2^*x_3^*| = \text{dist}_C(x_2^*,x_3^*)$. Let $\langle X_1, X_2, X_3\rangle$ be the dual of the optimal shortcut $x_1^*x_3^*$ such that $x_i^* \in \text{int}(X_i)$, $1 \le i \le 3$. Suppose that $2\text{dist}_C(b,x_3^*) + |x_2^*x_3^*| > \text{dist}_C(x_2^*,x_3^*)$. Thus, $\text{dist}_C(b,x_3^*) + |x_2^*x_3^*| > \frac{1}{2}(\text{dist}_C(x_2^*,x_3^*) + |x_2^*x_3^*|)$. Then we have two cases (see Figure 4).

Case 1. $\text{dist}_C(a,x_1^*) + |x_1^*x_2^*| \ge \frac{1}{2}(\text{dist}_C(x_1^*,x_2^*) + |x_1^*x_2^*|)$. Let $x_1^*x_2x_3$ be a node-0 shortcut of size 3 such that $x_i \in \text{int}(X_i)$, $2 \le i \le 3$, $x_3 \in \text{chain}_C(x_3^*,b)$, and $\text{dist}_C(b,x_3) + |x_2x_3| > \frac{1}{2}(\text{dist}_C(x_2,x_3) + |x_2x_3|)$. Thus,

$$\begin{aligned}\text{diam}(C \cup x_1^*x_3) &= \text{dist}_C(a,x_1^*) + |x_1^*x_3| + \text{dist}_C(x_3,b)\\ &< \text{dist}_C(a,x_1^*) + |x_1^*x_3^*| + \text{dist}_C(x_3^*,b)\\ &= \text{diam}(C \cup x_1^*x_3^*).\end{aligned}$$

This is a contradiction.

Case 2. $\text{dist}_C(a,x_1^*) + |x_1^*x_2^*| < \frac{1}{2}(\text{dist}_C(x_1^*,x_2^*) + |x_1^*x_2^*|)$. Let $x_1^*x_2x_3$ be a node-0 shortcut of size 3 such that $x_i \in \text{int}(X_i)$, $2 \le i \le 3$, $x_3 \in \text{chain}_C(x_3^*,b)$, $\text{dist}_C(a,x_1^*) + |x_1^*x_2| < \frac{1}{2}(\text{dist}_C(x_1^*,x_2) + |x_1^*x_2|)$, and $\text{dist}_C(b,x_3) + |x_2x_3| > \frac{1}{2}(\text{dist}_C(x_2,x_3) + |x_2x_3|)$. Thus,

$$\begin{aligned}\mathrm{diam}(C \cup x_1^* x_3) &= \tfrac{1}{2}(\mathrm{dist}_C(x_1^*, x_2) + |x_1^* x_2|) + |x_2 x_3| + \mathrm{dist}_C(x_3, b) \\ &< \tfrac{1}{2}(\mathrm{dist}_C(x_1^*, x_2^*) + |x_1^* x_2^*|) + |x_2^* x_3^*| + \mathrm{dist}_C(x_3^*, b) \\ &= \mathrm{diam}(C \cup x_1^* x_3^*).\end{aligned}$$

This is a contradiction.

Similarly, we can derive contradictions if $2\mathrm{dist}_C(b, x_3^*) + |x_2^* x_3^*| < \mathrm{dist}_C(x_2^*, x_3^*)$. Hence, $2\mathrm{dist}_C(b, x_3^*) + |x_2^* x_3^*| = \mathrm{dist}_C(x_2^*, x_3^*)$. Symmetrically, we know that $2\mathrm{dist}_C(a, x_1^*) + |x_1^* x_2^*| = \mathrm{dist}_C(x_1^*, x_2^*)$.

Definition 3. Let C be a simple chain with left end a and right end b, and xy be a shortcut of C with left end x and right end y. Let $\mathrm{chain}_C(a, x_1)$, $\mathrm{chain}_C(x_1, x_2)$, $\ldots$, $\mathrm{chain}_C(x_k, b)$ be a sequence of subchains of C partitioned by xy, where $\mathrm{dist}_C(a, x_1) < \mathrm{dist}_C(a, x_2) < \cdots < \mathrm{dist}_C(a, x_k)$. Let P_i, $1 \le i \le k-1$, be the simple polygon formed by the straight line segment $x_i x_{i+1}$ and the subchain $\mathrm{chain}_C(x_i, x_{i+1})$. The subchain $\mathrm{chain}_C(a, x_1)$ is considered as a degenerated polygon P_0 whose perimeter is $2\mathrm{dist}_C(a, x_1)$; and the subchain $\mathrm{chain}_C(x_k, b)$ is considered as a degenerated polygon P_k whose perimeter is $2\mathrm{dist}_C(b, x_k)$. The polygon sequence $(P_0, P_1, \ldots, P_k)$ is called the *partitioned polygon sequence* of xy. Let $x_0 = a$ and $x_{k+1} = b$. For each polygon P_i, $0 \le i \le k$, we call x_i the *left anchor* of P_i and x_{i+1} the *right anchor* of P_i. The perimeter of P_i is denoted by $\mathrm{peri}(P_i)$.

Theorem 9. *Let C be a simple chain that has an optimal node-0 shortcut S^* such that $|S^* \cap V(C)| = 0$ and $|S^* \cap C| \ge 4$. Let $\mathcal{P}$ be the partitioned polygon sequence of S^*. Then there are two polygons Q_1 and Q_2 in $\mathcal{P}$ listed from left to right, in which Q_1 and Q_2 are on the different sides of S^*, such that $\frac{1}{2}\mathrm{peri}(Q_1) + |x_1^r x_2^r| = \frac{1}{2}\mathrm{peri}(Q_2)$ or $\frac{1}{2}\mathrm{peri}(Q_1) = |x_1^l x_2^l| + \frac{1}{2}\mathrm{peri}(Q_2)$, where x_i^l is the left anchor and x_i^r is the right anchor of Q_i.*

6 Node-1 Shortcuts

Given a simple chain C, let $x^* y^*$ be an optimal shortcut of C. Since no three vertices of C are collinear, there are three cases for $x^* y^* \cap V(C)$: it is empty, it contains only one vertex, or it contains two vertices. In Section 5, we give necessary conditions for $x^* y^*$ when $x^* y^* \cap V(C) = \emptyset$. When $x^* y^* \cap V(C)$ contains two vertices, then it is easy to find $x^* y^*$. In this section, we consider necessary conditions for $x^* y^*$ when $x^* y^* \cap V(C)$ contains only one vertex of C.

Similar to Theorems 7 and 8, we can prove the following theorem.

Theorem 10. *Let $C = \mathrm{chain}(a, b)$ be a simple chain and $x^* y^*$ be an optimal shortcut of C with left end x^* and right end y^* such that $x^* y^* \cap V(C) = \{v\}$ and $x^* y^* \cap C = \{x^*, v, y^*\}$. Let a_v^* be the left pivot of v, b_v^* be the right pivot of v, and $a'a''$ and $b'b''$ be two edges of C such that $x^* \in \mathrm{int}(a'a'')$ and $y^* \in \mathrm{int}(b'b'')$. Then $x^* y^*$ has one of the following properties.*

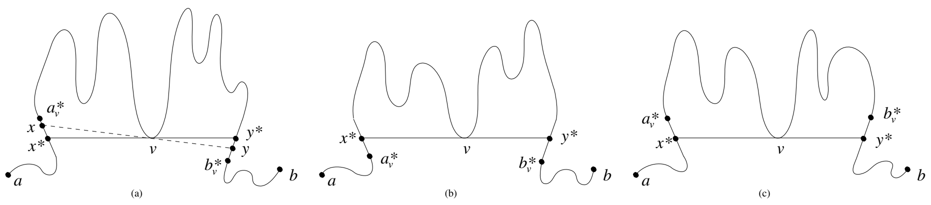


Fig. 5. (a) $x^* \in \text{chain}(a, a_v^*)$ and $y^* \in \text{chain}(v, b_v^*)$. (b) $x^*, y^* \in \text{chain}(a_v^*, b_v^*) - \{a_v^*, b_v^*\}$. (c) $x^* \in \text{chain}(a, a_v^*) - \{a_v^*\}$ and $y^* \in \text{chain}(b_v^*, b) - \{b_v^*\}$.

(i) *$x^* = a_v^*$ and $y^* = b_v^*$ (refer to Figure 4).*

(ii) *If $x^* \in \text{chain}(a, a_v^*)$ and $y^* \in \text{chain}(v, b_v^*)$, then $x^* = a_v^*$, or $y^* = b_v^*$, or $\text{dist}_C(a, x^*) = \text{dist}_C(b, y^*) + |y^*v|$, or $\text{dist}_C(b, y^*) = \frac{1}{2}(\text{dist}_C(v, x^*) + |x^*v|)$ (see Figure 5(a)).*

(iii) *If $x^* \in \text{chain}(a_v^*, v)$ and $y^* \in \text{chain}(b_v^*, b)$, then $x^* = a_v^*$, or $y^* = b_v^*$, or $\text{dist}_C(a, x^*) = \frac{1}{2}(\text{dist}_C(v, y^*) + |y^*v|)$, or $\text{dist}_C(b, y^*) = \text{dist}_C(a, x^*) + |x^*v|$.*

(iv) *If $x^*, y^* \in \text{chain}(a_v^*, b_v^*) - \{a_v^*, b_v^*\}$, then $\text{diam}(C \cup x^*y^*) = \min\{\text{dist}_C(a, x) + |xy| + \text{dist}_C(b, y)\colon x \in \text{int}(a'a''),\ y \in \text{int}(b'b'')$ and $xy \cap C = \{x, v, y\}\}$, or $\text{dist}_C(a, x^*) = \text{dist}_C(b, y^*) + |y^*v|$, or $\text{dist}_C(b, y^*) = \text{dist}_C(a, x^*) + |x^*v|$ (see Figure 5(b)).*

(v) *If $x^* \in \text{chain}(a, a_v^*) - \{a_v^*\}$ and $y^* \in \text{chain}(b_v^*, b) - \{b_v^*\}$, then $\text{diam}(C \cup x^*y^*) = \min\{\frac{1}{2}(\text{dist}_C(x, y) + |xy|)\colon x \in \text{int}(a'a''),\ y \in \text{int}(b'b'')$ and $xy \cap C = \{x, v, y\}\}$, or $\text{dist}_C(a, x^*) = \frac{1}{2}(\text{dist}_C(v, y^*) + |y^*v|)$, or $\text{dist}_C(b, y^*) = \frac{1}{2}(\text{dist}_C(v, x^*) + |x^*v|)$ (see Figure 5(c)).*

7 Shortcut Ratio

Let C be a simple chain with left end a and right end b, and xy be a shortcut of C with left end x and right end y. The *shortcut ratio* of xy is defined by $\gamma_C(xy) = \text{diam}(C \cup xy)/\text{dist}_C(a, b)$. In the case with no ambiguity, we use $\gamma(xy)$ without subscripts. In this section, we consider some upper bounds and lower bounds on the shortcut ratio.

It is easy to see that the shortcut ratio can be used to measure whether it is worth to build a shortcut in practice. It can also be used to measure the oscillation of chains. For example, let us consider two extreme cases. In the first case, the chain C is almost flat. Let C be a chain consisting of two edges ac and bc such that $|ac| = |bc| = 1$, and x^*y^* be an optimal shortcut of C. From Corollary 3, we know that

$$\gamma(x^*y^*) = \frac{\text{diam}(C \cup x^*y^*)}{\text{dist}_C(a, b)} = \frac{4 + \sqrt{2(1 - \cos\angle acb)}}{7 + \cos\angle acb}.$$

Thus, when $\angle acb$ approaches 180 degrees, the shortcut ratio approaches 1, which indicates that the chain is almost flat. We now consider the second case in which the oscillation frequency of the chain is very high. Let $(x_1, x_2, \ldots, x_n)$ be a simple chain such that $|x_1x_n| = 1$, $|x_1x_2| = |x_{n-1}x_n| = 1/2$, $|x_ix_{i+1}| = 1$ $(2 \leq i \leq n-2)$, and x_1x_n passes through all midpoints of edges x_ix_{i+1}, $2 \leq i \leq n-2$. Notice that

$$\gamma(x_1x_n) = \frac{\text{diam}(\text{chain}(x_1, x_n) \cup x_1x_n)}{\sum_{i=1}^{n-1} |x_ix_{i+1}|} < \frac{2}{n-2}.$$

Thus, when n approaches infinite, the shortcut ratio approaches 0, which indicates that the oscillation frequency of the chain is very high.

Theorem 11. *For a simple chain C with left end a and right end b, let xy be its optimal shortcut, and p and q be two points on C such that $|pq| = \max\{|uv| : u, v \in C\}$. Then*

$$\frac{|pq|}{\text{dist}_C(a,b)} \leq \gamma(xy) \leq \frac{1}{2} + \frac{|ab|}{2\text{dist}_C(a,b)}.$$

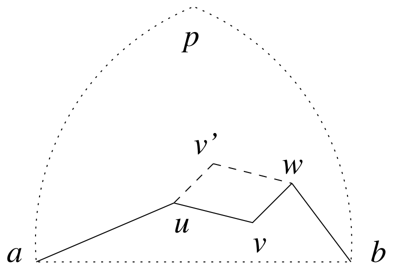


Fig. 6. Strongly monotonic chain chain(a, b)

For simple chains, we know that the shortcut ratio can be very close to 0. But for strongly monotonic chains, we can show that the shortcut ratio is bounded from $\frac{3}{2\pi}$.

Theorem 12. *Let C be a strongly monotonic chain with left end a and right end b, and xy be its optimal shortcut. Then $\gamma(xy) > \frac{3}{2\pi}$.*

Proof. We first consider the case that $C \cap ab = \{a, b\}$. Let p and q be the two intersection points of the two circles with the same radius $|ab|$ and different centers a and b, respectively. It follows from Theorem 2 that C is contained in the region bounded by the four circular arcs $\overset{\frown}{ap}$, $\overset{\frown}{pb}$, $\overset{\frown}{bq}$ and $\overset{\frown}{qa}$. Assume that C is contained in the region bounded by the segment ab and two arcs $\overset{\frown}{ap}$ and $\overset{\frown}{pb}$ (see Figure 6). Note that the length of each arc is $|ab|\pi/3$. Let P be the polygon formed by $C \cup ab$. If P is convex, since P must be contained in the region bounded by ab, $\overset{\frown}{ap}$ and $\overset{\frown}{pb}$, we know that $\text{dist}_C(a,b) < 2|ab|\pi/3$. Thus $\gamma(xy) = \frac{\text{diam}(C \cup xy)}{\text{dist}_C(a,b)} > \frac{|ab|}{2|ab|\pi/3} = \frac{3}{2\pi}$.

If P is not convex, let $v \in V(C)$ be a reflex vertex of P and uv and vw be two edges of C incident on v. Let v' be a point on the plane such that $uvwv'$ is a parallelogram (see Figure 6). Let C' be the chain obtained from C by replacing the subchain (u, v, w) by (u, v', w). Since $|uv'| = |vw|$ and $|v'w| = |uv|$, we can consider the edge sequence in C' as a permutation of the edge sequence in C. Since uv' and vw have the same slope, and $v'w$ and uv have the same slope, we can show that C' is also strongly monotonic.

Let P' be the polygon formed by $C' \cup ab$. If P' is not convex, we can repeat the above transformation until the resulted polygon is convex. Since the area of P' is strictly greater than that of P and C' can be considered as a permutation of C, the number of transformations is at most $(|V(C)| - 3)!$. Note that the length of the chain is always equal to $\text{dist}_C(a, b)$. Thus, $\gamma(xy) > \frac{3}{2\pi}$.

We now consider the case that $C \cap ab = \{x_1, x_2, \ldots, x_k\}$, where $x_1 = a$, $x_k = b$ and $\text{dist}_C(x_1, x_2) < \text{dist}_C(x_1, x_3) < \cdots < \text{dist}_C(x_1, x_k)$. For each subchain $\text{chain}_C(x_i, x_{i+1})$, $1 \leq i < k$, we know that $C \cap x_i x_{i+1} = \{x_i, x_{i+1}\}$. Thus, from the above, we have $\text{dist}_C(x_i, x_{i+1}) < \frac{2\pi}{3}|x_i x_{i+1}|$. Hence,

$$\text{dist}_C(a, b) = \sum_{i=1}^{k-1} \text{dist}_C(x_i, x_{i+1}) < \sum_{i=1}^{k-1} \frac{2\pi}{3}|x_i x_{i+1}| = \frac{2\pi}{3}|ab| \leq \frac{2\pi}{3}\text{diam}(C \cup xy).$$

Therefore, $\gamma(xy) > \frac{3}{2\pi}$

8 Algorithms

In any realistic computation model, it may be impossible to evaluate the diameter exactly because it may contain square roots that can be irrational numbers. In this section, we present three approximation algorithms for computing optimal shortcuts of general chains.

Algorithm 1
Input: a chain C and a positive number ϵ.
Output: a shortcut of C.

1. For each pair of vertices in C, find the maximal shortcut that contains these two vertices.
2. For each $i = 0, 1, \ldots, \lceil \frac{180}{\epsilon} \rceil$, we sweep a line whose slope is $\tan(i\epsilon)$ to find a shortcut X such that $\text{diam}(C \cup X)$ is minimum among all shortcuts with slope $\tan(i\epsilon)$.
3. Find the shortcut S such that $\text{diam}(C \cup S)$ is minimum among all shortcuts found in Steps 1 and 2. Output S.

Theorem 13. *Given a chain C with n vertices and a positive number ϵ, Algorithm 1 can find a shortcut S of C in $O(n^2 + \frac{1}{\epsilon} n \log n)$ time such that* $\text{diam}(C \cup S) \leq (1 + \frac{1}{2}\epsilon)\text{diam}(C \cup S^*)$, *where S^* is an optimal shortcut of C.*

From Theorem 13, we know that Algorithm 1 is a fully polynomial-time approximation scheme, that is, FPTAS. If the length of a chain is not very long, we have a better approximation algorithm.

Algorithm 2
Input: a chain C and a positive number ϵ.
Output: a shortcut of C.

1. Add a set U of points on C to partition it into pieces such that the length of each piece is at most ϵ.
2. For each pair of points in $V(C) \cup U$, find the maximal shortcut that contains these two points.
3. Find the shortcut S such that $\mathrm{diam}(C \cup S)$ is minimum among all maximal shortcuts found in Step 2. Output S.

Theorem 14. *Given a chain C with n vertices and a positive number ϵ, Algorithm 2 can find a shortcut S of C in $O((n + \frac{1}{\epsilon}|C|)^2 n)$ time such that* $\mathrm{diam}(C \cup S) \leq \mathrm{diam}(C \cup S^*) + 6\epsilon$*, where S^* is an optimal shortcut of C and $|C|$ is the length of C.*

If the length of a chain is too long but the perimeter of its convex hull is not long, we can modify algorithm 2 by adding points on the convex hull of the chain as follows.

Algorithm 3
Input: a chain C and a positive number ϵ.
Output: a shortcut of C.

1. Find the convex hull $CH(C)$ of $V(C)$. Add a set U of points on $CH(C)$ to partition it into pieces such that the length of each piece is at most ϵ.
2. For each pair of points $u, v \in U$, find the maximal shortcut of C that is contained in the segment uv.
3. For each pair of vertices in C, find the maximal shortcut that contains these two vertices.
4. Find the shortcut S such that $\mathrm{diam}(C \cup S)$ is minimum among all maximal shortcuts found in Steps 2 and 3. Output S.

Theorem 15. *Given a chain C with n vertices and a positive number ϵ, Algorithm 3 can find a shortcut S of C in $O((\frac{1}{\epsilon}p)^2 n)$ time such that* $\mathrm{diam}(C \cup S) \leq \mathrm{diam}(C \cup S^*) + O(\epsilon)$*, where S^* is an optimal shortcut of C and p is the perimeter of $CH(C)$.*

9 Concluding Remarks

In Section 8, we present three approximation algorithms for computing optimal shortcuts of general chains. For simple chains, we can use the necessary conditions of optimal shortcuts in Sections 5 and 6 and the following two lemmas to improve the efficiency of these algorithms.

Let $L_g(h)$ be a straight line with slope g and y-intercept h. When g is fixed, and h changes, $L_g(h)$ is a function of h, which represents a sweep line with slope g. Given a chain C, let $S(L_g(h))$ denote the maximal shortcut induced by $L_g(h)$. Thus, $\mathrm{diam}(C \cup S(L_g(h)))$ is a function of h. For simple chains we can show that this function is unimodal.

Lemma 4. *Given a simple chain C and a slope g, let $L_g(h)$ be a sweep line with y-intercept h and $S(L_g(h))$ be the maximal shortcut induced by $L_g(h)$. Then* $\mathrm{diam}(C \cup S(L_g(h)))$ *is a unimodal function of h.*

Lemma 5. *Let C be a simple chain with left end a and right end b, and x^*y^* be an optimal shortcut of C with left end x^* and right end y^*. Let a' and b' be two points on C such that* $\mathrm{dist}_C(a,a') = \mathrm{dist}_C(b,b') = \frac{1}{3}\mathrm{dist}_C(a,b)$. *Then* $x^* \in \mathrm{chain}(a,a')$ *and* $y^* \in \mathrm{chain}(b,b')$.

A number of interesting questions remain open:

1. Given a chain, how to add k shortcuts such that the new graph has the minimum diameter?
2. Given a Euclidean graph, how to add a shortcut (or k shortcuts) such that the new graph has the minimum diameter?
3. Given a Euclidean graph, how to add a k-chain to the graph such that the new graph has the minimum diameter?

Acknowledgments. Part of this work was done when the author visited the Chinese University of Hong Kong. The author would like to thank Leizhen Cai for introducing the minimum diameter problem and having valuable discussions.

References

1. Bentley, J.L., Ottmann, T.A.: Algorithms for reporting and counting geometric intersections. IEEE Trans. Comput. C 28, 643–647 (1979)
2. Brown, K.Q.: Comments on Algorithms for reporting and counting geometric intersections. IEEE Trans. Comput. C-30, 147–148 (1981)
3. Cai, L.: Personal communication
4. Gudmundsson, J., Haverkort, H., Park, S., Shin, C., Wolff, A.: Facility location and the geometric minimum-diameter spanning tree. Computational Geometry: Theory and Applications 27, 87–106 (2004)
5. Ho, J., Lee, D., Chang, C., Wong, C.: Minimum diameter spanning trees and related problems. SIAM Journal on Computing 20, 987–997 (1991)
6. Preparata, F.P., Shamos, M.I.: Computational Geometry - An Introduction. Springer, Heidelberg (1985)
7. Spriggs, M., Keil, J.M., Bespamyatnikh, S., Segal, M., Snoeyink, J.: Computing a $(1+\epsilon)$-approximate geometric minimum-diameter spanning tree. Algorithmica 38, 577–589 (2004)

List Dynamic Coloring of Sparse Graphs

Seog-Jin Kim[⋆] and Won-Jin Park[⋆⋆]

Department of Mathematics Education, Konkuk University, Seoul, South Korea

Abstract. A *dynamic coloring* of a graph G is a proper coloring of the vertex set $V(G)$ such that each vertex neighborhood of size at least 2 receives at least two distinct colors. The *list dynamic chromatic number* $ch_d(G)$ of G is the least integer k such that for every list assignment of size k to each vertex of G, there is a dynamic coloring of G such that each vertex is colored by a color from its list. We proved that $ch_d(G) \leq 4$ if $\text{Mad}(G) < \frac{8}{3}$ where $\text{Mad}(G)$ is the maximum average degree of G. And $ch_d(G) \leq 4$ if G is a planar graph of girth at least 7. Both results are sharp. In addition, we show that $ch_d(G) \leq 6$ for every planar graph G.

Keywords: Dynamic coloring, list coloring, planar graph, maximum average degree.

1 Introduction

All graphs in this paper are finite, undirected and simple. A *dynamic coloring* of a graph G is a proper coloring of the vertex set $V(G)$ such that each vertex neighborhood of size at least 2 receives at least two distinct colors.. A dynamic k-coloring of a graph is a dynamic coloring by k colors. The smallest integer k such that G has a dynamic k-coloring is called the *dynamic chromatic number* $\chi_d(G)$ of G.

The relationship between $\chi(G)$ and $\chi_d(G)$ has been studied in several papers (see [2], [5], [6], [9]). The gap $\chi_d(G) - \chi(G)$ could be infinitely large for some graphs, but for some graphs $\chi_d(G) - \chi(G)$ is small. It is interesting problem to study which graphs have small value of $\chi_d(G) - \chi(G)$.

Fan, Lai, and Chen [4] showed that $\chi_d(G) \leq 5$ if G is a planar graph, and it is conjectured that $\chi_d(G) \leq 4$ if G is a planar graph other than C_5. However it must be a very difficult conjecture by considering the Four color problem. Hence it would be interesting to study which planar graphs have dynamic chromatic number at most 4. Meng et al. [8] showed that the dynamic chromatic number of Pseudo-Halin graphs, which is a class of planar graphs, is at most 4.

As an analogue of Grötzsch's theorem for planar graph, one can wonder if there is a constant integer k such that every planar graph of girth at least k has dynamic chromatic number at most 3 or 4, where the *girth* of a graph G is the length of the smallest cycle of G. Since $\chi_d(C_k) = 4$ where $k \geq 7$ and k is not

[⋆] skim12@konkuk.ac.kr

[⋆⋆] Department of Mathematics, Seoul National University, eotp11@math.snu.ac.kr multiple of 3, it is a natural question what is the smallest integer k that every planar graph of girth at least k has dynamic chromatic number at most 4, if it exists.

In this paper, we find conditions on maximum average degree and girth to guarantee that $\chi_d(G) \leq 4$. The *maximum average degree*, denoted $\mathrm{Mad}(G)$, is $\max_{H\subseteq G} \frac{2|E(H)|}{|V(H)|}$. Since $\mathrm{Mad}(G) < \frac{2g(G)}{g(G)-2}$ for every planar graph G, studying dynamic coloring in terms of maximum average degree is a natural approach.

For every vertex $v \in V(G)$, let $L(v)$ denote a list of colors available at v. An L-coloring is a proper coloring ϕ such that $\phi(v) \in L(v)$ for every vertex $v \in V(G)$. A graph G is k-choosable if it has an L-coloring whenever all lists have size at least k. The list chromatic number $\chi_l(G)$ of G is the least integer k such that G is k-choosable. A dynamic L-coloring is a dynamic coloring of G such that each vertex is colored by a color from its list. A graph G is called dynamic k-choosable if it has a dynamic L-coloring whenever all lists have size at least k. The *list dynamic chromatic number* $ch_d(G)$ of G is the least integer k such that G is dynamic k-choosable.

Note that $\chi(G) \leq \chi_d(G) \leq ch_d(G)$ for every graph G. Hence the upper bounds on $ch_d(G)$ are also upper bounds on $\chi_d(G)$. In this paper, we find conditions on maximum average degree and girth to guarantee that $ch_d(G) \leq 4$. We show that $ch_d(G) \leq 4$ if $\mathrm{Mad}(G) < \frac{8}{3}$. The maximum average degree condition is tight, since there is a graph G such that $\mathrm{Mad}(G) = \frac{8}{3}$ but $\chi_d(G) = 5$. The result implies that $ch_d(G) \leq 4$ if the girth of the planar graph is at least 8 by the inequality $\mathrm{Mad}(G) < \frac{2g(G)}{g(G)-2}$. We improve the girth bound further for planar graph. We show that list dynamic chromatic number of planar graph G is at most 4 if the girth of G is at least 7. Again the condition on girth is tight since there is a planar graph H whose girth is 6 but $ch_d(H) = 5$.

In addition, we find an upper bound of list dynamic chromatic number of planar graphs. We show that $ch_d(G) \leq 6$ for every planar graph G.

2 List Dynamic Coloring of Sparse Graphs

Theorem 1. *The list dynamic chromatic number $ch_d(G)$ is at most 4 if* $\mathrm{Mad}(G) < \frac{8}{3}$. *This is sharp.*

Proof. Suppose that the theorem is false. Let G be a counterexample. That is, each vertex of G has list assignment L of size 4 and $\mathrm{Mad}(G) < \frac{8}{3}$, but G has no dynamic coloring from the lists. We assume that G has the smallest number of vertices among such counterexamples.

Claim 1. *Minimal counterexample G cannot have two adjacent 2-vertices.*

Proof. Suppose that there are adjacent 2-vertices u, v. Without loss of generality, we may assume that the other neighbor w of u other than v has degree at least 3 in G since we can select such adjacent 2-vertices. Let v_1 be the other neighbor of v other than u. Let $H = G - \{u, v\}$. Note that $d_H(w) \geq 2$. By induction hypothesis, H has a dynamic L-coloring c by the minimality of G. Color v

by a color p in $L(v) \setminus \{c(w), c(v_1), c(v_2)\}$ where v_2 is a neighbor of v_1 other than v. And then color u by a color q in $L(u) \setminus \{c(w), c(v), c(v_1)\}$. Note that we can select such colors p and q since $|L(v)| \geq 4$ and $|L(u)| \geq 4$. Now G has a dynamic L-coloring. This contradiction completes the proof of Claim 1.

Claim 2. *Minimal counterexample G cannot have adjacent 2-vertex and 3-vertex.*

Proof. Suppose that 2-vertex v is adjacent to a 3-vertex u. Let w_1 and w_2 be the other two neighbors of u other than v, and w_3 is the other neighbor of v other than u. Then we have following two cases.

Case 1: $d(w_i) \geq 3$ for $i = 1, 2$.

Let $H = G - \{u, v\}$. Then by induction hypothesis, H has a dynamic L-coloring c. Color the vertex u by a color x in $L(u) \setminus \{c(w_1), c(w_2), c(w_3)\}$. And then color v by a color in $L(u) \setminus \{c(w_1), c(w_3), x\}$ if $c(w_1) = c(w_2)$. Otherwise color v by a color in $L(v) \setminus \{c(w_3), x\}$. Then G has a L-dynamic coloring.

Case 2: $d(w_1) = 2$ or $d(w_2) = 2$.

Without loss of generality, we may assume that $d(w_1) = 2$. Let z_1 be the other neighbor of w_1 other than u. By Claim 1, $d(z_1) \geq 3$. Here w_2 may be a 2-vertex. In this case, let z_2 be the other neighbor of w_2. Again we know that $d(z_2) \geq 3$.

Let $H = G - \{u, v, w_1\}$ ($H = G - \{u, v, w_1, w_2\}$ if $d(w_2) = 2$). Then by induction hypothesis, H has a dynamic L-coloring c. Color u by a color in $L(u) \setminus \{c(z_1), c(w_2), c(w_3)\}$. If $d(w_2) = 2$, then color u by a color in $L(u) \setminus \{c(z_1), c(z_2), c(w_3)\}$. And then color the remaining 2-vertices w_1, v (w_1, w_2, v if $d(w_2) = 2$) greedily. Then G has a dynamic L-coloring. This contradiction completes the proof of Claim 2.

Now we use discharging method to show that the minimal counterexample G has maximum average degree at least $8/3$, which implies that there is no counterexample.

We have the following discharging rules.

Discharging Rule: If v is adjacent to 2-vertex w where $d(v) \geq 4$, then v gives charge $\frac{d(v)-8/3}{d(v)}$ to w.

By the discharging rule, we show that every vertex has new charge $d^*(v) \geq 8/3$. If $d(v) = 2$, then all neighbors of v have degree at least 4. Hence v receives at least $2 \cdot (\frac{4-8/3}{4})$. Hence $d^*(v) \geq 2 + 2/3 = 8/3$. If $d(v) = 3$, then $d^*(v) = d(v) = 3$. Now if $d(v) \geq 4$, then $d^*(v) \geq d(v) - d(v) \cdot (\frac{d(v)-8/3}{d(v)}) = 8/3$.

Hence $\text{Mad}(G) \geq 8/3$. This contradiction completes the proof of Theorem 1. □

Remark 1. The $\text{Mad}(G)$ condition is tight. Let G be the graph obtained by subdividing of the edges of K_5. Then $\text{Mad}(G) = 8/3$, but $\chi_d(G) = 5$. Hence $ch_d(G) = 5$.

Theorem 2. *The list dynamic chromatic number $ch_d(G)$ is at most 4 if G is a planar graph with girth at least 7.*

Proof. Suppose that the theorem is false. Let G be a minimal counterexample. Note that G has no cut-edge. Now Claim 1 and Claim 2 also hold for G. Hence if v is a 2-vertex in G, then the both neighbors of v have degree at least 4.

Define charge $\phi(v) = d(v) - 4$ if $v \in V(G)$ and $\phi(F) = d(F) - 4$ if F is a face. Then

$$\sum_{v \in V(G)} \phi(v) + \sum_{F \text{ face}} \phi(F) = -8$$

Next we will distribute the charge ϕ between faces and vertices by the following discharging rules.

Rule 1: Each face gives charge 1 to each incident 2-vertex.

Rule 2: Each face gives charge 1/3 to each incident 3-vertex.

After discharging, it is easy to show that $\phi^*(v) \geq 0$ for all vertices. Now we will show that $\phi^*(F) \geq 0$ for all faces. Let F be a face and let C be a cycle whose edge set is the same as the face F. Let n_2, n_3, and n_4^+ be the number of vertices of 2-vertices, 3-vertices, and vertices of degree at least 4 on the cycle C, respectively. By Claim 1 and Claim 2, we know that $n_4^+ \geq n_2$ if $|C| \geq 8$ and $n_4^+ \geq n_2 + 1$ if $|C| = 7$. Hence, if $|C| \geq 8$, then

$$|C| = n_2 + n_3 + n_4^+ \geq 2n_2 + n_3.$$

Thus $n_2 \leq \frac{1}{2}\big(|C| - n_3\big)$. Hence

$$\phi^*(F) \geq (|C| - 4) - \frac{1}{2}\big(|C| - n_3\big) - \frac{1}{3}n_3 = \frac{1}{2}\big(|C| - 8\big) + \frac{1}{6}n_3 \geq 0.$$

On the other hand, if $|C| = 7$, then

$$|C| = n_2 + n_3 + n_4^+ \geq 2n_2 + 1 + n_3.$$

Thus $n_2 \leq \frac{1}{2}\big(|C| - 1 - n_3\big)$. Hence

$$\phi^*(F) \geq (|C| - 4) - \frac{1}{2}\big(|C| - 1 - n_3\big) - \frac{1}{3}n_3 = \frac{1}{2}\big(|C| - 7\big) + \frac{1}{6}n_3 \geq 0.$$

Hence $\phi^*(F) \geq 0$ for every face F. Therefore $\phi^*(w) \geq 0$ for each $w \in V(G) \cup F(G)$. This implies that

$$-8 = \sum_{v \in V(G)} \phi(v) + \sum_{F \text{ face}} \phi(F) = \sum_{v \in V(G)} \phi^*(v) + \sum_{F \text{ face}} \phi^*(F) \geq 0.$$

This contradiction complete the proof of the theorem. □

It is a natural question whether there is a constant integer k such that $\chi_d(G) \leq 3$ for every connected planar graph G with $g(G) \geq k$ and $\Delta(G) \geq 3$. For an answer for this question, we will show that for any positive integer k where $k \equiv 1 \pmod 3$ there is a planar graph G of girth k and arbitrary large maximum degree $\Delta(G)$ such that $\chi_d(G) = 4$.

Example 1. For each integer k where $k = 3s + 1$ for some integer s, let G be a planar graph consisted of $(\Delta - 1)$ cycle of length k with common edge uv. The maximum degree of G is Δ and the girth of G is k. Note that $d_G(u) = \Delta = d_G(v)$ and the other vertices have degree 2.

Now suppose that G has a dynamic 3-coloring ϕ. Then without loss of generality, we may assume that $\phi(u) = 1$ and $\phi(v) = 2$. Then u has a neighbor u_1 whose color is 3 to satisfy dynamic coloring. Let C be the cycle that contains the vertex u_1. Since $|C| = 3s + 1$, the path $P = V(C) - \{u, v\}$ has $3(s - 1) + 2$ vertices. Since the cycle C must satisfy dynamic coloring, the color of the path P should be repeated by $3, 2, 1, 3, 2, 1 \ldots$, from the vertex u_1. But, since the path has $3(s - 1) + 2$ vertices, we can check easily that the cycle C cannot complete dynamic coloring. This contradiction implies that $\chi_d(G) > 3$.

Remark 2. In general, we can extend the result of Theorem 1 for arbitrary integer $k \geq 4$. We can show that $ch_d(G) \leq k$ if $\text{Mad}(G) < \frac{4k}{k+2}$ for $k \geq 4$ by an similar argument in Theorem 1, and this result is also sharp. Note that the graph G obtained from the complete graph K_{k+1} by subdividing every edge has $\text{Mad}(G) = \frac{4k}{k+2}$, but $\chi_d(G) = k + 1$.

3 List Dynamic Chromatic Number of Planar Graphs

In this section, we will show that $ch_d(G) \leq 6$ for every planar graph G. For an edge $e = v_1v_2$ in G, let d_1, d_2 denote the degrees of the two endpoints v_1 and v_2, respectively, and d_1^*, d_2^* denote the length of the two faces adjacent to the edge v_1v_2. The edge distribution of e is defined by $\phi(e) = \frac{1}{d_1} + \frac{1}{d_2} + \frac{1}{d_1^*} + \frac{1}{d_2^*} - 1$. We have the following theorem that is essentially same as Lebesque Theorem [7].

Theorem 3. *([4]) If G is a planar graph, then* $\displaystyle\sum_{e \in E(G)} \phi(e) = 2$.

The *edge configuration* of an edge is the 4-tuple (x_1, x_2, x_3, x_4) such that $x_1 \leq x_2 \leq x_3 \leq x_4$ that are obtained by ordering d_1, d_2, d_1^*, d_2^*. For convenience, we use (x_1, x_2, x_3, S) where S is a set of numbers, meaning that x_4 can be any integer in S. The following configuration is obtained in [4].

Lemma 1. *([4]) Let G be a planar graph with $\delta(G) \geq 3$. Then there is an edge whose configuration is one of the four cases.*
(a) $(3, 3, 3, [3, \infty])$
(b) $(3, 3, 4, [4, 11])$
(c) $(3, 3, 5, [5, 7])$
(d) $(3, 4, 4, [4, 5])$.

Theorem 4. $ch_d(G) \leq 6$ *for every planar graph G.*

Proof. Suppose that the theorem does not hold, and assume that G is a counterexample with $|V(G)|$ minimized and let L be the list assignment of size 6 such that G has no dynamic list coloring from L.

Claim 3. *$\delta(G) \geq 3$.*

If G has a vertex v of degree 1, then $H = G - v$ is dynamic L-coloring ϕ. Let u be the neighbor of v in G. Color v by a color c in $L(v) \setminus \{\phi(u), \phi(u_1)\}$ where u_1 is a vertex in $N(u)$ different from v. Then G has a dynamic L-coloring. It is a contradiction.

If G has a vertex v of degree 2, then put $H = G - v + xy$ where x, y are the neighbors of v in G. Then by the hypothesis, H has a dynamic L-coloring ϕ. Color v by a color c in $L(v) \setminus \{\phi(x), \phi(y), \phi(x_1), \phi(y_1)\}$ where x_1 is a neighbor of x other than y and y_1 is a neighbor of y other than x in G. Then G has a dynamic L-coloring. It is a contradiction.

Hence we may assume that $\delta(G) \geq 3$. We prove the following two claims to complete the proof.

Claim 4. *If $\delta(G) \geq 3$, then G does not have a vertex v such that $d_G(v) \leq 5$ and v has a pair of adjacent neighbors.*

Let x, y be a pair of adjacent neighbors of v. Let $H = G - v$. Since G is a minimal counterexample, H has a dynamic L-coloring ϕ. Since x, y are adjacent in H, at least two colors are used in $N_G(v)$. Since $\delta(H) \geq 2$, each neighbor of v is adjacent to at least two different colors. Color v by a color c in $L(v) \setminus \phi(N_G(v))$, then G has a dynamic L-coloring. This contradiction completes the proof of Claim 4.

Claim 5. *If $\delta(G) \geq 3$, then G does not have a pair of adjacent vertices u and v such that $d_G(u) = 3$ and $d_G(v) \leq 5$.*

If $N_G(u) \cap N_G(v) \neq \emptyset$, then it is done by Claim 4. If $N_G(u) \cap N_G(v) = \emptyset$, then let $H = G - \{u, v\}$. Then since G is a minimal counterexample, H has a dynamic L-coloring ϕ. Since $\delta(H) \geq 2$, the each of neighbors of u or v is adjacent to at least two different colors. Color v by a color c_1 in $L(v) \setminus \big(\phi\big(N_G(v) \setminus \{u\}\big) \cup \{\phi(u_1)\}\big)$ where u_1 is a neighbor of u other than v. Also color u by a color c_2 in $L(u) \setminus \big(\phi\big(N_G(u) \setminus \{v\}\big) \cup \{c_1, \phi(v_1)\}\big)$ where v_1 is a neighbor of v other than u. Since $d_G(u) = 3$ and $d_G(v) \leq 5$, such colors c_1, c_2 are available. Hence G has a dynamic L-coloring. It is a contradiction. Hence Claim 5 is proved.

By Lemma 1, there is an edge v_1v_2 whose configuration is one of the following four cases. We may assume that $d(v_1) \leq d(v_2)$ and $d_1^* \leq d_2^*$.

Case 1: $x_1 = x_2 = x_3 = 3$ and $x_4 \in [3, \infty]$.

In this case, the degree of v_1 is 3 and a pair of neighbors of v_1 are adjacent. It cannot happen by Claim 4.

Case 2: $x_1 = x_2 = 3$ and $x_3 = 4$ and $4 \leq x_4 \leq 11$.

If $d_1 = d_2 = 3$, then it is a contradiction by Claim 5. Otherwise, $d(v_1) \leq 4$ and a pair of neighbors of v_1 are adjacent. It cannot happen by Claim 4.

Case 3: $x_1 = x_2 = 3$ and $x_3 = 5$ and $5 \leq x_4 \leq 7$.

If $d_1 = d_2 = 3$, then it is a contradiction by Claim 5. Otherwise, $d(v_1) \leq 5$ and a pair of neighbors of v_1 are adjacent. It cannot happen by Claim 4.

Case 4: $x_1 = 3$, $x_2 = x_3 = 4$ and $4 \leq x_4 \leq 5$.

If $d_1^* = 3$, then it is a contradiction by Claim 4. Otherwise, $d(v_1) = 3$ and $d(v_2) = 4$. It cannot happen by Claim 5.

Therefore none of the above four cases happens. It is a contradiction. Hence there is no counterexample. Thus $ch_d(G) \leq 6$ for every planar graph G. □

We have found a sharp girth condition to guarantee that $ch_d(G) \leq 4$ for every planar graph G. But, we do not know a sharp girth condition to have that $\chi_d(G) \leq 4$ for every planar graph G. It would be an interesting problem to check if $\chi_d(G) \leq 4$ for every planar graph G of girth at least 4. We showed that $ch_d(G) \leq 6$ for every planar graph G. But, it is not known whether the upper bound $ch_d(G) \leq 6$ is tight or not. Hence it would be interesting to answer the following question.

Question 1. Is it true that $ch_d(G) \leq 5$ for every planar graph?

Acknowledgements. We thank Professor A.V. Kostochka and Professor D.B. West for helpful discussion.

References

1. Akbari, S., Ghanbari, M., Jahanbekam, S.: On the list dynamic coloring of graphs. Discrete Applied Mathematics 157, 3005–3007 (2009)
2. Alishahi, M.: On Dynamic coloring of graphs (2009) (manuscript)
3. Esperet, L.: Dynamic list coloring of bipartite graphs, to appear at Discrete Applied Mathematics
4. Fan, S., Lai, H.-J., Chen, Y.: Conditional coloring for planar graphs and graphs of higher genus (manuscript)
5. Lai, H.-J., Lin, J., Montgomery, B., Poon, H.: Upper bounds of dynamic chromatic number. Ars Combinatoria 68, 193–197 (2003)
6. Lai, H.-J., Lin, J., Montgomery, B., Shui, T., Fan, S.: Conditional colorings of graphs. Discrete Math. 306, 1997–2004 (2006)
7. Lebesque, H.: Quelques conséquences simples de la formule d'Euler. J. de Math. 9, Sér. 19, 27–43 (1940)
8. Meng, X., Miao, L., Su, B., Li, R.: The Dynamic Coloring Numbers of Pseudo-Halin Graphs. Ars Combinatoria 79, 3–9 (2006)
9. Montgomery, B.: Dynamic coloring of graphs, Ph.D. dissertation, West Virginia University (2001)

Further Improvement on Maximum Independent Set in Degree-4 Graphs*

Mingyu Xiao[1] and Hiroshi Nagamochi[2]

[1] School of Computer Science and Engineering,
University of Electronic Science and Technology of China, China
myxiao@gmail.com
[2] Department of Applied Mathematics and Physics,
Graduate School of Informatics, Kyoto University, Japan
nag@amp.i.kyoto-u.ac.jp

Abstract. We present a simple algorithm for the maximum independent set problem in an n-vertex graph with degree bounded by 4, which runs in $O^*(1.1526^n)$ time and improves all previous algorithms for this problem. In this paper, we use the "Measure and Conquer method" to analyze the running time bound, and use some good reduction and branching rules to avoid tedious checking on a large number of local structures.

Keywords: Exact Algorithm, Independent Set, Measure and Conquer.

1 Introduction

The *maximum independent set* problem (MIS), to find a maximum set of vertices in a graph such that there is no edge between any two vertices in the set, is not only a basic problem introduced in Garey and Johnson's work [10] on NP-completeness, but also one of the most important problems in the line of research on worst-case analysis of algorithms for NP-hard optimization problems. Since Tarjan and Trojanowski [16] published the first nontrivial $O^*(2^{n/3})$-time algorithm in 1977, the bound of the running time to exactly solve the problem has been improved frequently [11,14,15,7,12,3]. One of the most important results among them is that due to Fomin *et al.* [7], in which they use a new method called "Measure and Conquer" to analyze simple algorithms. By using this method together with other techniques, recently Kneis *et al.* [12] and Bourgeois *et al.* [3] improved the running time bound to $O^*(1.2132^n)$ and $O^*(1.2127^n)$ respectively. Up to till now, these are the best published results for MIS.

One of the most important subcases to solve MIS is the problem in low-degree graphs. Since we can simply branch on a high-degree vertex by including it into the independent set or excluding it from the independent set, and then reduce the graph greatly in the subbranches, sometimes the problem in degree-i graphs (graphs with maximum degree i) for small i will become the bottleneck for solving the problem in general graphs. Bourgeois *et al.* [3] also used a bottom-up method

* Partially supported by National Natural Science Foundation of China under the Grant 60903007.

to get improvements on MIS. In this method, the running time bounds for MIS in degree-3 graphs (MIS3) and MIS in degree-4 graphs (MIS4) will directly affect the algorithms and running time bounds for the problem in other degree bounded graphs and general graphs. Motivated by these, researchers have great interests in designing fast algorithms for MIS3 and MIS4. For MIS3, we quote the $O^*(1.1259^n)$-time algorithm by Beigel [1], the $O^*(1.1225^n)$-time algorithm by Fomin and Høie [8], the $O^*(1.1120^n)$-time algorithm by Fürer [9], the $O^*(1.1034^n)$-time algorithm by Xiao *et al.* [18], the $O^*(1.0892^n)$-time algorithm by Razgon [13], the $O^*(1.0885^n)$-time algorithm by Xiao [19], and finally the $O^*(1.0854^n)$-time algorithm by Bourgeois *et al.* [2]. For MIS4, MIS in degree-5 graphs (MIS5) and MIS in degree-6 graphs (MIS6), the best previous results are $O^*(1.1571^n)$, $O^*(1.1918^n)$ and $O^*(1.2071^n)$ respectively [3], which are designed by using a bottom-up method based on a fast algorithm for MIS3. In this paper, we will design a simple algorithm for MIS4 by using "Measure and Conquer," which runs in $O^*(1.1526^n)$ time and improves all previous algorithms for this problem.

Most fast exponential-time algorithms are based on a branch-and-reduce paradigm, which contains two main steps. We first check whether we can get partial solution and reduce the current problem directly according the *reduction rules*, and then branch the problem instance into several smaller instances according to the *branching rules*. To scale the size of the instance, we may need to use a parameter, such as the number of vertices or edges for graph problems, as a measure of the size of the instance. By bounding the size of the search tree to a function of the parameter, we will get a running time bound relating to the parameter for the problem. For MIS, we branch on the current graph G into several graphs G_1, $G_2, \ldots, G_l$ such that the parameter w_i of each graph G_i is less than the parameter w of graph G, and a maximum independent set in G can be found in polynomial time if a maximum independent set in each of the l graphs G_1, $G_2, \ldots, G_l$ is known. Usually, G_i $(i = 1, 2, \ldots, l)$ are obtained by deleting some vertices in G. We can build up a search tree according to our branching rules, and the exponential part of the running time of the algorithm corresponds to the size of the search tree. The running time analysis leads to a linear recurrence for each node in the search tree that can be solved by using standard techniques. By letting $C(w)$ denote the worst-case size of the search tree when the parameter of graph G is w, we get the recurrence relation $C(w) \leq \sum_{i=1}^{l} C(w - w'_i)$, where $w - w'_i = w_i$. Solving the recurrence, we get $C(w) = [\alpha(w'_1, w'_2, \ldots, w'_l)]^w$, where $\alpha(w'_1, w'_2, \ldots, w'_l)$ is the largest root of the function $f(x) = 1 - \sum_{i=1}^{l} x^{-w'_i}$. As for the measure (the parameter w), we should guarantee that when parameter $w \leq 0$ the problem can be solved directly and in each step (applying reduction rules or branching rules) the parameter will not increase. A natural measure is the number of vertices or edges in the graph. Note that to get fast algorithms by this method, we hope that w'_i $(i = 1, 2, \ldots, l)$ in the above recurrence are as large as possible. To get large values of them, some papers check a large number of local structures of the graph and get numerous branching rules. However, the "Measure and Conquer" method tries to improve the recurrences in another way. In this method, we set a weight to each vertex in the graph according to the degree of the vertex (usually vertices of the same degree receive the same weight) and use the sum of the weights in the graph as the measure. Note that when a vertex v is deleted, we may reduce the measure not only from v but also from the neighbors of v (the degrees of the neighbors will decrease by 1). Compared to traditional measures, the weighted measure may catch more structure information of the graph and can get further improvement without modifying the algorithms. Currently, the best exact algorithms for many NP-hard problems are designed by this method. However, we should choose a good weight for the vertices, which is an important step in this method. To do this, we may need to solve a quasiconvex program. In this paper we also use the branch-and-reduce paradigm and the "Measure and Conquer" method to design our algorithm.

The rest of the paper is organized as follows. Section 2 gives the notation that may be used in the paper. Sections 3 and 4 introduce the reduction rules and branching rules, respectively. Section 5 presents our simple algorithm. Section 6 analyzes the running time bound. Finally Section 7 makes some concluding remarks.

2 Notation System

Given a graph $G = (V, E)$, the number vertices of degree i in the graph is denoted by n_i and the total number of vertices in the graph is denoted by n. For a vertex v in a graph, $d(v)$ is the degree of v, $N(v)$ the set of all neighbors of v, $N[v] = N(v) \cup \{v\}$ the set of vertices with distance at most 1 from v, and $N_2(v)$ the set of vertices with distance exactly 2 from v, and $N_2[v] = N_2(v) \cup N[v]$. We may also use $N(V')$ to denote the neighbors of a set V' of vertices, i.e., $N(V') = \cup_{v \in V'} N(v) - V'$. A *line graph* of graph G is the graph whose vertices are corresponding to the edges of G, and two vertices are adjacent if and only if the corresponding two edges sharing a same endpoint in G. In our algorithm, when we remove a set of vertices, we also remove all the edges that are incident on it. Throughout the paper we use a modified O notation that suppresses all polynomially bounded factors. For two functions f and g, we write $f(n) = O^*(g(n))$ if $f(n) = g(n)poly(n)$, where $poly(n)$ is a polynomial in n.

3 Reduction Rules

Reduction rules are used to reduce the size of instances of the problem directly before applying the branching rules. Reduction rules will not exponentially increase the size of our search tree. Furthermore, the reduction operations will reduce some special local structures of the graph, and then the branching rules can apply effectively in the resulted graphs. Most of the reduction rules used here are well-known in the literature. Some of them are newly introduced.

Folding degree-1 vertices
Folding a degree-1 vertex v means removing v together with its neighbor u from the graph.

Folding degree-2 vertices
Folding a degree-2 vertex v (with two neighbors a and b) means

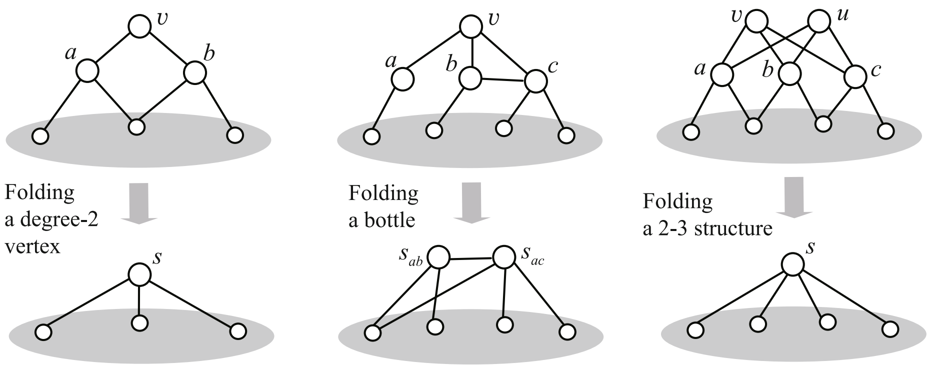


Fig. 1. Illustrations of folding operations

(*a*) *removing* v, a *and* b *from the graph, when* a *and* b *are adjacent.*
(*b*) *contracting* v, a *and* b *into a single vertex* s, *when* a *and* b *are nonadjacent.*

Figure 1 illustrates the operation in case (b) of folding a degree-2 vertex. We may also reduce a special case of degree-3 vertices. Let v be a degree-3 vertex, and a, b, c the three neighbors of v. If two neighbors of v, say b and c, are adjacent, then we say that the four vertices compose a *bottle* and denote it by a-v-$\{b, c\}$. Specially, when a is a degree-3 vertex, we say that bottle a-v-$\{b, c\}$ is a *weak bottle.* For a bottle, we have the following reduction rule.

Folding bottles
In the operation of folding a bottle a-v-$\{b, c\}$, *we add two adjacent vertices* s_{ab} *and* s_{ac}, *add edges between* s_{ab} *and each vertices in* $N(a) \cup N(b)$, *add edges between* s_{ac} *and each vertices in* $N(a) \cup N(c)$, *and delete* $N[v]$ *from the graph.*

Figure 1 illustrates the operation of folding a bottle. Let $\alpha(G)$ denote the size of a maximum independent set of graph G and $G^{\star}(v)$ the graph after folding a degree-1 or degree-2 vertex v or a bottle a-v-$\{b, c\}$ in G. Then we have the following lemma.

Lemma 1. *For any degree-*1 *or degree-*2 *vertex* v *or a bottle* a-v-$\{b, c\}$ *in graph* G, *we have* $\alpha(G) = 1 + \alpha(G^{\star}(v))$.

The correctness of the reduction rules has been discussed in many references [4,7]. In fact, folding a bottle is a special case of a reduction rule introduced in [7]. We give a new name of the local structure just for the convenience of the analysis. Note that in our algorithm, we will just reduce weak bottles and keep some other bottles, because applying this reduction rule to non-weak bottles increases our measure (defined in Section 6), which is unexpected in our algorithm.

In our algorithm, we will also use the following reduction rule. If two independent degree-3 vertices v and u have three common neighbors a, b and c, then we say that the five vertices compose a 2-3 *structure* (see Figure 1), and denote it by $\{v, u\}$-$\{a, b, c\}$.

Folding 2-3 structures
Folding a 2-3 structure A-B means contracting $A \cup B$ into a singe vertex and deleting parallel edges and self-loops from the graph.

Lemma 2. *If graph G has a 2-3 structure, then $\alpha(G) = 2 + \alpha(G^\star)$, where $G^\star$ is the graph obtained from G by folding a 2-3 structure in G.*

The above lemma is proved in [19]. In fact, it is a special case of the crown reduction introduced in [5].

Dominance
We say that a vertex u dominates another vertex v if $N[u] \subseteq N[v]$.

Lemma 3. *If a vertex v is dominated by any other vertices in graph G, then*

$$\alpha(G) = \alpha(G - \{v\}).$$

Line graphs
If graph G is a line graph of graph G', we find a maximum independent set of G directly by finding a maximum matching in G' and taking the corresponding vertex set in G as the solution.

Not every graph is a line graph. There are several good methods to check whether a graph is a line graph or not, which depend on characterizations of line graphs [17]. In this paper, we only need to check whether a graph is a line graph of a 3-regular graph, which can be easily done (note that a graph is a line graph of a 3-regular graph if and only if the graph has only degree-4 vertices and each of them is contained in two edge-disjoint triangles).

Bipartite graphs
If graph G is a bipartite graph, we find a maximum independent set of G in polynomial time by Hungarian Algorithm.

Definition 1. *A graph is called a* reduced graph*, if it contains none of degree-1 vertices, degree-2 vertices, weak bottles, 2-3 structures and dominated vertices, and has no connected component which is a line graph of a 3-regular graph or a bipartite graph.*

4 Branching Rules

In the algorithm, we may branch on a vertex v of maximum degree by including it into the independent set or excluding it from the independent set. That is, in the first branch we will delete $N[v]$ from the graph and in the second branch we will delete v from the graph. Besides this branching rule, we also use other two branching rules, *branching on a 4-cycle* and *branching on a bottle*, first introduced in [19].

For four vertices a, b, c and d in graph G, we say that $abcd$ is a *4-cycle* in G if there are four edges ab, bc, cd and da.

Lemma 4. *Let abcd be a* 4*-cycle in graph* G*. For any independent set* S *in* G*, either* $a, c \notin S$ *or* $b, d \notin S$*.*

Proof. Since any independent set contains at most 2 vertices in a 4-cycle and the two vertices cannot be adjacent, we know the lemma holds. ∎

Based on Lemma 4, we get the following branching rule.

Branching on a 4**-cycle**
Branching on a 4*-cycle abcd means branching by either excluding* a *and* c *from the independent set or excluding* b *and* d *from the independent set.*

We have introduced a reduction rule that can reduce bottles. However, we just use the rule to reduce weak bottles. For other kinds of bottles, we may use some branching rules to deal with them.

Lemma 5. *Let* a*-*v*-*$\{b, c\}$ *be a bottle in graph* G*. Then there is a maximum independent set* S *in* G *such that either* $a \in S$ *or* $v \in S$*.*

Proof. If a is not in a maximum independent set, we can directly remove a from the graph. In the remaining graph v becomes a degree-2 vertex and the two neighbors of it are adjacent. In this case, there is a maximum independent set that contains v. ∎

Based on Lemma 5, we get the following branching rule.

Branching on a bottle
Branching on a bottle a*-*v*-*$\{b, c\}$ *means branching by either including* a *in the independent set or including* v *in the independent set.*

5 The Algorithm for MIS4

We call a degree-4 vertex a *good degree-*4 *vertex* if it is not contained in two edge-disjoint triangles. Our algorithm for MIS4 is described in Figure 2.

6 The Analysis

We will use the measure and conquer method to analyze the running time bound of our algorithm. We set a weight to each vertex in the graph according to the degree of the vertex, $w : \mathbb{Z}_+ \to \mathbb{R}_+$ (where $\mathbb{Z}_+$ and $\mathbb{R}_+$ denote the sets of nonnegative integers and nonnegative reals, respectively): we denote by w_i the weight $w(v)$ of a vertex v of each degree $i \geq 0$. Then we adopt $w = \sum_i w_i n_i$ as the measure of the graph. We will show that when $w \leq 0$, the problem can be solved in polynomial time. Note that we allow the weight w_i of a vertex to be a number greater than 1. We predecide the value of w_i for $i \neq 3$ by the following formula:

$$w_i = \begin{cases} 0 & \text{if } i \leq 2, \\ 1 & \text{if } i = 4, \\ 1 + w_3 & \text{if } i = 5, \\ 2 + (i-6)(1-w_3) & \text{if } i \geq 6. \end{cases}$$

```
Input: A graph $G$.
Output: The size of a maximum independent set in $G$.

 1. If {There is a degree-1 or degree-2 vertex $v$ or a weak bottle $a$-$v$-$\{b,c\}$},
    return $1 + MIS4(G^\star(v))$.
 2. If {There is a 2-3 structure}, return $2 + MIS4(G^\star)$.
 3. If {$\exists v, u \in V$: $N[u] \subseteq N[v]$}, return $MIS4(G - \{v\})$.
 4. If {The graph has a component $P$ that has at most 15 vertices, or is the
    line graph of a 3-regular graph or a bipartite graph}, return $t + MIS4(G - P)$,
    where $t$ is the size of a maximum independent set in $P$.
 5. If {There is a vertex of degree $\geq 5$}, pick up a vertex $v$ of maximum
    degree, and return $\max\{MIS4(G - \{v\}), 1 + MIS4(G - N[v])\}$.
 6. If {There is a bottle $a$-$v$-$\{b,c\}$ such that one of $b$ and $c$ is a degree-4
    vertex}, return $\max\{1 + MIS4(G - N[a]), 1 + MIS4(G - N[v])\}$.
 7. If{There is a 4-cycle $abcd$ that contains a degree-4 vertex}, return
    $\max\{MIS4(G - \{a,c\}), MIS4(G - \{b,d\})\}$.
 8. If {There are two adjacent degree-4 vertices}, pick up a good degree-
    4 vertex $v$ that is adjacent to at least one degree-4 vertex, and return
    $\max\{MIS4(G - \{v\}), 1 + MIS4(G - N[v])\}$.
 9. If {There are still some degree-4 vertices}, pick up a degree-4 vertex $v$ such
    that the number of degree-3 vertices in $N_2(v)$ is maximized, and return
    $\max\{MIS4(G - \{v\}), 1 + MIS4(G - N[v])\}$.
10. Else {The graph is a 3-regular graph}, we use a fast algorithm for MIS3
    to solve the problem and return a solution.

Note: With a few modifications, the algorithm can provide a maximum inde-
pendent set itself.
```

Fig. 2. The Algorithm $MIS4(G)$

We only need to assign the value to w_3, which decides the value of w_i for all other i's. In the analysis, by solving a finite dimensional quasiconvex program, we will get that $w_3 = 0.5908$. Note that when measure $w = 0$, the graph has only degree-0, degree-1 and degree-2 vertices and the maximum independent set problem can be solved in linear time. This is the boundary condition of our search tree. Initially, the graph has no vertex of degree > 4. We also have that $w_i \leq 1$ for $i = 0, 1, 2, 3, 4$. Then at the beginning the measure w is not greater than the number n of vertices. If we can get a running time bound related to measure w, then we can also get a running time bound related to n. To get a running time bound of the algorithm, we show that the measure w will not increase when we apply the reduction rules and analyze how much we can reduce the measure w in each branching step in the algorithm.

6.1 Preliminaries

Lemma 6. *The measure w of a graph will not increase, if we apply the reduction rules of folding a degree-1 or degree-2 vertex, a weak bottle or a 2-3 structure,* *or removing a dominated vertex, a connected component of a line graph of a 3-regular graph or a bipartite graph.*

It is a simple matter to verify the lemma by straightforward calculations. Note that folding non-weak bottles may increase w, which is not contained in our reduction rules.

Next, we focus on the analysis for branching rules. We will use $\varDelta w_i$ to denote $w_i - w_{i-1}$ for $i \geq 1$. Then

$$\varDelta w_i = \begin{cases} 0 & \text{if } i = 1 \text{ or } 2, \\ w_3 & \text{if } i = 3 \text{ or } 5, \\ 1 - w_3 & \text{if } i = 4 \text{ or } \geq 6. \end{cases}$$

In our algorithm, to simplify the analysis, we will also require that $2/3 \geq w_3 \geq 0.5$. This restriction is the 1-st constraint in our quasiconvex program to solve the best value of w_3. Then we have that $w_3 \geq 1 - w_3$, $w_3 \leq 2(1 - w_3)$ and $\varDelta w_i \geq 1 - w_3$ for $i \geq 3$. These properties will be used frequently in our analysis. We may not claim this every time. For example, we consider the change of weights when we remove a set X of vertices from a graph with minimum degree at least 3. If there are $3k + i$ ($i \in \{-1, 0, 1\}$) edges between X and $V - X$, then we observe that the total weight in the remaining set $V - X$ decreases at least by $kw_3 + (1 - w_3)\delta$, where $\delta = 1$ when $i = 1$, or $\delta = 0$ otherwise.

6.2 Step 5

After Step 4, the graph is a reduced graph where the minimum degree is at least 3. In Step 5, the algorithm will branch on a vertex v of maximum degree by excluding it from the independent set or including it into the independent set. In the first branch, we will delete v from the graph. In the second branch, we will delete $N[v]$ from the graph. We use $\varDelta_{out}(v)$ and $\varDelta_{in}(v)$ to denote the amount of w being reduced in the corresponding two branchings respectively. Assume that v is of degree $d \geq 5$ and has d_i neighbors of degree i. Then $d = \sum_{i=3}^{d} d_i$. To analyze how much w can be reduced in each branch, we consider two cases.

When $d \geq 6$, we get:

$$\varDelta_{out}(v) = w_d + \sum_{i=3}^{d} d_i \varDelta w_i \geq w_d + d(1 - w_3) \geq w_6 + 6(1 - w_3) = 8 - 6w_3,$$

and

$$\varDelta_{in}(v) = w_d + \sum_{i=3}^{d} d_i w_i \geq w_6 + 6w_3 = 2 + 6w_3.$$

Let $C(w)$ denote the worst-case size of the search tree when the parameter of the graph is w. We have the following recurrence

$$C(w) = C(w - \varDelta_{out}(v)) + C(w - \varDelta_{in}(v)) \leq C(w - (8 - 6w_3)) + C(w - (2 + 6w_3)).$$

This recurrence will generate the 2-nd constraint in our quasiconvex program to solve the best value of w_3.

When $d = 5$, it holds $d = d_5 + d_4 + d_3$, and we get:

$$\Delta_{out}(v) = w_5 + d_5 w_3 + d_4(1 - w_3) + d_3 w_3 = 1 + 6w_3 + (1 - 2w_3)d_4.$$

In the branch where v is included into the independent set, we will reduce all vertices in $N[v]$ and reduce the degree of the vertices in $N_2(v)$. For each vertex $v' \in N(v)$, there is an edge between v' and a vertex in $N_2(v)$, otherwise v' would dominate v. Hence there are at least $|N(v)| \geq 5$ edges between $N[v]$ and $N_2(v)$, and deleting $N[v]$ reduces w by more than $2w_3$ from $N_2(v)$. We get

$$\begin{aligned}\Delta_{in}(v) &\geq w_d + \textstyle\sum_{i=3}^{5} d_i w_i + 2w_3 \geq w_5 + d_4 + (5 - d_4)w_3 + 2w_3\\ &= 1 + 8w_3 + (1 - w_3)d_4.\end{aligned}$$

Therefore, we get the recurrence for the case $d = 5$:

$$\begin{aligned}C(w) &= C(w - \Delta_{out}(v)) + C(w - \Delta_{in}(v))\\ &\leq C(w - (1 + 6w_3 + (1 - 2w_3)d_4)) + C(w - (1 + 8w_3 + (1 - w_3)d_4)).\end{aligned}$$

By replacing d_4 with $0, 1, 2, 3, 4$ and 5 in the above relation, we can get six different recurrences. These six recurrences will generate the 3-rd to the 8-th constraints in our quasiconvex program to solve the best value of w_3.

6.3 Step 6

In this step, the graph is a reduced graph where there is no weak bottle and each vertex is of degree 3 or 4. If the graph contains a bottle a-v-$\{b, c\}$ such that at least one of b and c, say b, is a degree-4 vertex, then the algorithm will branch on it by removing $N[a]$ or $N[v]$. Note that the degree of a is 4, since there is no weak bottle in the graph. Also there is no edge between a and b or a and c, otherwise v would dominate b or c.

First, we look at the branch where $N[a]$ is removed. There are at least 5 edges between $N[a]$ and $N_2(a)$, otherwise a neighbor of a would dominate a. We also see that $N_2(a)$ contains at least three vertices, two of which are b and c, otherwise the graph would not be a reduced graph. If $|N[a]| = 3$, then after removing $N[a]$, we can reduce w by at least $1 + 2(1 - w_3) = 3 - 2w_3$ from $N_2(a)$ (1 from b, $1 - w_3$ from c and $1 - w_3$ from another vertex in $N_2(v)$). If $|N[a]| > 3$, we can reduce w by at least $5(1 - w_3)$ from $N_2(a)$ after removing $N[a]$. Note that $5(1 - w_3) \geq 3 - 2w_3$. We can also reduce w by at least $1 + 4w_3$ from $N[a]$ itself. Then in this branch, we can always reduce w by at least $1 + 4w_3 + 3 - 2w_3 = 4 + 2w_3$.

For the branch where $N[v]$ is removed, we consider two cases: the degree of c is 3 or 4. When c is a degree-3 vertex, there are 6 edges between $N(v)$ and $N_2(v)$. Note that b and c have no common neighbor in $N_2(v)$, otherwise c would dominate b. Hence it is impossible to create a degree-0 vertex after removing $N[v]$. Then after removing $N[v]$, we can reduce w by at least $3w_3$ from $N_2(v)$ (note that $w_3 \leq 2(1 - w_3)$). Then in this branch, totally we can reduce

w by at least $2 + 2w_3 + 3w_3 = 2 + 5w_3$. When c is a degree-4 vertex, there are 7 edges between $N(v)$ and $N_2(v)$. Note that b and c can have at most one common neighbor in $N_2(v)$, otherwise c would dominate b. Hence it is impossible to create two degree-0 vertices after removing $N[v]$. Then after removing $N[v]$, we can reduce w by at least $3w_3$ from $N_2(v)$ too. Totally we can reduce w by at least $3 + w_3 + 3w_3 = 3 + 4w_3 > 2 + 5w_3$.

In this step, we can always branch with the following recurrence

$$C(w) \leq C(w - (4 + 2w_3)) + C(w - (2 + 5w_3)). \qquad (1)$$

Note that after this step, the graph has no triangle that contains both degree-3 and degree-4 vertices.

6.4 Step 7

In this step, we will branch on 4-cycles that contain at least one degree-4 vertex. Without loss of generality, we assume that the algorithm will branch on 4-cycle $abcd$, where a is a degree-4 vertex.

According to the branching rule, our algorithm will branch by removing either $\{a, c\}$ or $\{b, d\}$ from the graph. We distinguish the following five cases.

Case 1: There is only one vertex a of degree 4 in the 4-cycle where a and c (b and d) are not adjacent, otherwise a (b) would dominate c (d). We assume that a' and a'' are the third and fourth neighbors of a, b' is the third neighbor of b, c' is the third neighbor of c, and d' is the third neighbor of d (see Figure 3(a) for an illustration). Note that $b' \neq d'$, otherwise $\{b, d\}$-$\{a, c, b' = d'\}$ would be a 2-3 structure. Also $\{b', d'\} \cap \{a', a'', c'\} = \emptyset$, otherwise there would be a weak bottle.

In the branch where $\{a, c\}$ are removed, b and d will become degree-1 vertices. The algorithm will apply the reduction rules to reduce degree-1 vertices immediately. Then b' and d' will be removed. Totally, at least 6 vertices a, b, c, d, b'

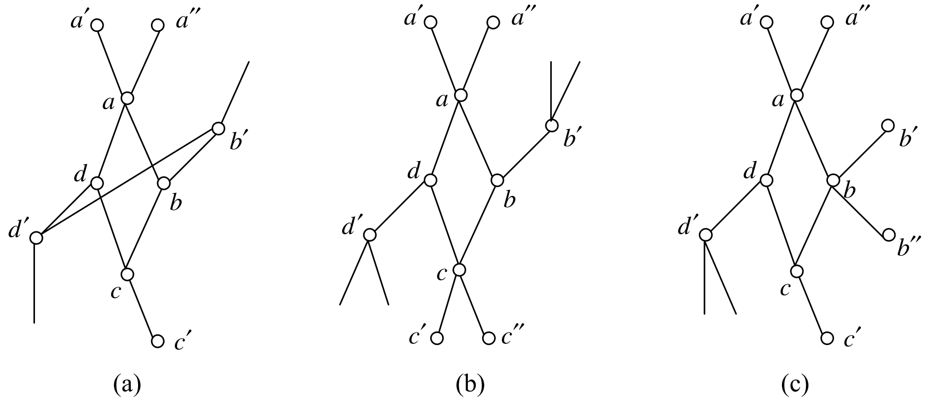


Fig. 3. Branching on 4-cycles

and d' are removed from the graph. There are also at least 5 edges between $V' = \{a, b, c, d, b', d'\}$ and $V - V'$ (there may not be 7 edges when b' and d' are adjacent). We consider how much we can reduce w from $V - V'$ after removing V'. If $|N(V')| \geq 3$, we can reduce w by at least $w_3 + 2(1 - w_3) = 2 - w_3$. If $|N(V')| = 2$, we can either reduce w by $1 + (1 - w_3)$ (at least one vertex in $N(V')$ is a degree-4 vertex) or reduce w by at least $2w_3$ (both vertices in $N(V')$ are degree-3 vertices) together with some degree-1 vertex created. For the later case, we can further reduce w by at least $1 - w_3$ by folding degree-1 vertices, and together can reduce w by at least $2w_3 + (1 - w_3) = 1 + w_3$ from $V - V'$. For any case, we can reduce w by at least $2 - w_3$ from $V - V'$ after removing V' (note the $w_3 \geq 1 - w_3$). Then in this branch we can reduce w by at least

$$1 + 5w_3 + 2 - w_3 = 3 + 4w_3.$$

In the branch where $\{b, d\}$ are removed, c will become a degree-1 vertex and we will further remove c' from the graph. Note that there are at least 6 edges between $V' = \{b, c, d, c'\}$ and $V - V'$. After removing V', we can reduce w by 1 from vertex a and reduce w by at least $2w_3$ from $V - V' \cup \{a\}$ (note that it is impossible to create a degree-0 vertex after removing V', otherwise the graph is not a reduced graph). Then in this branch we can reduce w by at least

$$4w_3 + 1 + 2w_3 = 1 + 6w_3.$$

For this case, we can always branch with the following recurrence

$$C(w) \leq C(w - (3 + 4w_3)) + C(w - (1 + 6w_3)). \tag{2}$$

Case 2: There are two degree-4 vertices a and c in the 4-cycle. Let b' and d' be the third neighbor of b and of d respectively. Note that $b' \neq d'$ holds, b' (d') is not adjacent to a or c, and a and c are not adjacent to each other, since the graph is a reduced graph that has no triangle containing both degree-3 and degree-4 vertices. See Figure 3(b) for an illustration of this case.

It is easy to see that in the branch where $\{b, d\}$ are removed, we can reduce w by $2w_3$ from b and d, by 2 from a and c, and by at least $2(1 - w_3)$ from b' and d'. Totally, we can reduce w by $2w_3 + 2 + 2(1 - w_3) = 4$. In the branch where $\{a, c\}$ are removed, b and d become degree-1 vertices and we will also further remove b' and d'. Let $V' = \{a, b, c, d, b', d'\}$. We consider how much we can reduce w from $V - V'$ after removing V'. Note that there are at least 6 edges between V' and $V - V'$ (b' and d' may be adjacent) and $|N(V')| \geq 4$. Then we can reduce w by at least $w_3 + 3(1 - w_3)$ from $V - V'$. Totally we can reduce w by at least $2 + 4w_3 + w_3 + 3(1 - w_3) = 5 + 2w_3$. For this case, we can branch with

$$C(w) \leq C(w - 4) + C(w - (5 + 2w_3)). \tag{3}$$

Case 3: There are two degree-4 vertices a and b (or a and d) in the 4-cycle. Assume without loss of generality that a and b are the degree-4 vertices in the cycle where c (d) is not adjacent to a (b) since it dominates no other vertex. Define a', a'', b',

c' and d' as in Case 1, and let b'' be the fourth neighbor of b (see Figure 3(c) for an illustration). Since the graph has no weak bottle, c' (d') is different from any of b', b'' (a', a'') whereas $d' \in \{b', b''\}$ and $\{a', a''\} \cap \{b', b'', c'\} \neq \emptyset$ and $c' = d'$ are possible. We look at the branch where $\{a, c\}$ are removed. Vertex d will become a degree-1 vertex and we will further fold the degree-1 vertex d by removing d and its third neighbor d'. Then in this branch we will remove $N[d]$. We consider how much we can reduce w from $V - N[d]$ after removing $N[d]$. Note that at most one pair of vertices in $N(d)$ can be adjacent (for this case, $c' = d'$). There are at least 5 edges between $N(d)$ and $N_2(d)$. Since $a', a'', b \in N_2(d)$, it holds $|N_2(d)| \geq 3$. If $|N_2(d)| \geq 4$, then we can reduce w by at least $1 + 3(1 - w_3)$ from $N_2(d)$ (1 from b and $3(1 - w_3)$ from the other vertices in $N_2(d)$). For the case of $|N_2(d)| = 3$, we may only guarantee that w will be reduced by at least $1 + 2(1 - w_3)$ from $N_2(d)$. However, when the worst case happens, a' or $a'' \in N_2(d)$ will become a degree-1 vertex. By further reducing degree-1 vertices, we can further reduce w by at least $1 - w_3$ (note that the graph has more than 15 vertices). Therefore, we still can reduce w by at least $1 + 3(1 - w_3)$ from $V - N[d]$. In the branch where $\{a, c\}$ are removed, we can always reduce w by at least $1 + 3w_3 + 1 + 3(1 - w_3) = 5$. This also holds for the branch where $\{b, d\}$ are removed. We can branch with

$$C(w) \leq 2C(w - 5). \tag{4}$$

Case 4: There are three degree-4 vertices in the 4-cycle. Without loss of generality, we assume that the three degree-4 vertices are a, b and c. Note that a and c are not adjacent and the third neighbor d' of d is not adjacent to a or c. In the branch where $\{a, c\}$ are removed, vertex d becomes a degree-1 vertex and we further fold d by removing $\{d, d'\}$. Then we can reduce w by 3 from $\{a, b, c\}$, at least $2w_3$ from $\{d, d'\}$, and at least $2w_3$ from $V - \{a, b, c, d, d'\}$. Totally, we can reduce w by at least $3 + 4w_3$. In the other branch, we can reduce w by at least $3 + w_3 + 2w_3 = 3 + 3w_3$. We get recurrence

$$C(w) \leq C(w - (3 + 4w_3)) + C(w - (3 + 3w_3)). \tag{5}$$

Case 5: All the vertices in the 4-cycle are degree-4 vertices. Note that for this case, a and c (also, b and d) may be adjacent to each other. It is easy to see that in each branch we can reduce w by 4 from $\{a, b, c, d\}$ and at least $w_3 + (1 - w_3) = 1$ from $V - \{a, b, c, d\}$. We get the same recurrence as (4).

Note that after this step, no degree-4 vertex is contained in a 4-cycle.

6.5 Step 8

First of all, we show that there is always a good degree-4 vertex adjacent to another degree-4 vertex if the graph has two adjacent degree-4 vertices in this step. Note that if all vertices in a connected component are degree-4 vertices and none is a good degree-4 vertex, then this component is the line graph of a 3-regular graph, which will be reduced in Step 4. Otherwise, there is a degree-4 vertex v adjacent to both degree-3 and degree-4 vertices. If v is contained in two edge-disjoint triangles, then there is triangle that contains both degree-3 and degree-4 vertices, which will form a weak bottle or a bottle satisfying the condition in Step 6. Then v is a good degree-4 vertex.

Let v be the good degree-4 vertex selected in this step. We will branch by either deleting v from the graph or deleting $N[v]$ from the graph. We distinguish the following five cases according to the number of degree-4 vertices in $N(v)$. Note that there is at least one degree-4 vertex in $N(v)$.

Case 1: There is only one degree-4 vertex in $N(v)$. Then $|N_2(v)| = 9$, otherwise there would be a 4-cycle containing a degree-4 vertex or a triangle containing both degree-3 and degree-4 vertices. The only case of the local structure is showed in Figure 4. In the branch where v is removed, we can reduce w by $1 + 3w_3 + (1 - w_3) = 2 + 2w_3$. In the branch where $N[v]$ is removed, we can reduce w by at least $2 + 3w_3 + 9(1 - w_3) = 11 - 6w_3$. We get recurrence

$$C(w) \leq C(w - (2 + 2w_3)) + C(w - (11 - 6w_3)). \tag{6}$$

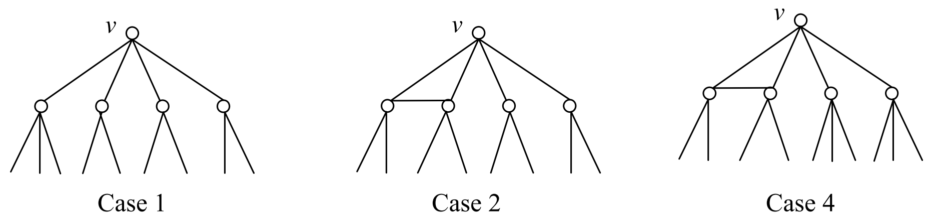


Fig. 4. Some cases of branching on a degree-4 vertex

Case 2: There are two degree-4 vertices in $N(v)$. We have two subcases: the two degree-4 vertices are adjacent or not. It is easy to see that the case of adjacent will cover the other case. We assume that the two degree-4 vertices are adjacent to each other (see Figure 4). Then $|N_2(v)| = 8$. When v is removed, we can reduce w by $1 + 2w_3 + 2(1 - w_3) = 3$. When $N[v]$ is removed, we can reduce w by at least $3 + 2w_3 + 8(1 - w_3) = 11 - 6w_3$. We get recurrence

$$C(w) \leq C(w - 3) + C(w - (11 - 6w_3)). \tag{7}$$

Case 3: There are three degree-4 vertices in $N(v)$. There is also at most one edge with two endpoints in $N(v)$. Then $|N_2(v)| \geq 9$. When v is removed, we can reduce w by $1 + w_3 + 3(1 - w_3) = 4 - 2w_3$. When $N[v]$ is removed, we can reduce w by at least $4 + w_3 + 9(1 - w_3) = 13 - 8w_3$. We get recurrence

$$C(w) \leq C(w - (4 - 2w_3)) + C(w - (13 - 8w_3)). \tag{8}$$

Case 4: All vertices in $N(v)$ are degree-4 vertices. Since v is a good degree-4 vertex that is not contained in any 4-cycles. There is also at most one edge with two endpoints in $N(v)$ (see Figure 4). For this case, $|N_2(v)| \geq 10$. When v is removed, we can reduce w by $1+4(1-w_3)=5-4w_3$. When $N[v]$ is removed, we can reduce w by at least $5+10(1-w_3)=15-10w_3$. We get recurrence

$$C(w) \le C(w-(5-4w_3)) + C(w-(15-10w_3)). \tag{9}$$

6.6 Step 9

In this step, the set of degree-4 vertices is an independent set. Let v be a degree-4 vertex selected in this step. Then the neighbors of v are four degree-3 vertices. We show that there is at least one degree-3 vertex in $N_2(v)$. Assume to the contrary that for each vertex v', $N_2(v')$ contains only degree-4 vertices. Then the graph is a bipartite graph with one side of degree-3 vertices and the other degree-4 vertices, which will be reduced by our reduction rules. Now we branch on v. In the branching where v is removed, we can reduce w by $1+4w_3$ directly. Note that $|N_2(v)|=8$ (v is not contained in any 3-cycle or 4-cycle). Then in the branching where $N[v]$ is removed we can reduce w by at least $7(1-w_3)+w_3=7-6w_3$ from $N_2(v)$ and reduce w by $1+4w_3$ from $N[v]$ itself. Totally we can reduce w by at least $8-2w_3$. We get recurrence

$$C(w) \le C(w-(1+4w_3)) + C(w-(8-2w_3)). \tag{10}$$

6.7 Step 10

It is easy to see that if none of the first 9 steps can be executed, the graph is a 3-regular graph. We will use a fast algorithm for MIS3 to solve it. Here we use the $O^*(1.0854^n)$-time algorithm by Bourgeois *et al.* [2], and then this step will not be the bottleneck of our algorithm. For this step, we get running time bound

$$C(w) = O(1.0854^{\frac{w}{w_3}}), \tag{11}$$

which will generate the last constraint in our quasiconvex program.

6.8 Putting All Together

Recurrences (1) to (10) generate the 9-th to 18-th constraint in our quasiconvex program. Consider an assignment of w_3 satisfying $0.5 \le w_3 \le 2/3$. By solving the i-th recurrence ($2 \le i \le 18$), we will get that $C(w) \le (\alpha_i(w_3))^w$. We need to choose a value of w_3 such that $\max(\alpha_2(w_3), \alpha_3(w_3), \cdots, \alpha_{18}(w_3), 1.0854^{\frac{1}{w_3}})$ is minimized. By solving this quasiconvex program according to the method introduced in [6], we get a running time bound of $O(1.1526^w)$ by setting $w_3 = 0.5908$ for our problem. Now the bottlenecks are (7) and (10), i.e., the 15-th and 18-th constraints in our quasiconvex program.

Theorem 1. *A maximum independent set in a degree-*4 *graph of* n *vertices can be found in* $O^*(1.1526^n)$ *time.*

7 Concluding Remarks

In this paper, we have designed a fast algorithm for the maximum independent set problem in graphs with degree bounded by 4, which is analyzed by the "Measure and Conquer" method. Different from most previous "Measure and Conquer" algorithms, our algorithm allows the weight of vertices greater than 1 and do not require $\Delta w_i \geq \Delta w_{i+1}$. We get rid of these two frequently used assumptions to simplify the analysis. In this paper, we have clearly listed out all constraints in our quasiconvex program and pointed out the bottlenecks of the algorithm. However, it is hard for most previous "Measure and Conquer" algorithms to do that, since there are a huge number of constraints in their quasiconvex programs.

References

1. Beigel, R.: Finding maximum independent sets in sparse and general graphs. In: SODA 1999, pp. 856–857. ACM Press, New York (1999)
2. Bourgeois, N., Escoffier, B., Paschos, V.T., van Rooij, J.M.M.: Maximum independent set in graphs of average degree at most three in $\mathcal{O}(1.08537^n)$. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 373–384. Springer, Heidelberg (2010)
3. Bourgeois, N., Escoffier, B., Paschos, V.T., van Rooij, J.M.M.: A bottom-up method and fast algorithms for max independent set. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 62–73. Springer, Heidelberg (2010)
4. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theoretical Computer Science 411(40-42), 3736–3756 (2010)
5. Chor, B., Fellows, M., Juedes, D.W.: Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 257–269. Springer, Heidelberg (2004)
6. Eppstein, D.: Quasiconvex analysis of backtracking algorithms. In: SODA, pp. 781–790. ACM Press, New York (2004)
7. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In: SODA, pp. 18–25. ACM Press, New York (2006)
8. Fomin, F.V., Høie, K.: Pathwidth of cubic graphs and exact algorithms. Inf. Process. Lett. 97(5), 191–196 (2006)
9. Fürer, M.: A faster algorithm for finding maximum independent sets in sparse graphs. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 491–501. Springer, Heidelberg (2006)
10. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness. Freeman, San Francisco (1979)
11. Jian, T.: An $O(2^{0.304n})$ algorithm for solving maximum independent set problem. IEEE Transactions on Computers 35(9), 847–851 (1986)
12. Kneis, J., Langer, A., Rossmanith, P.: A fine-grained analysis of a simple independent set algorithm. In: Kannan, R., Kumar, K.N. (eds.) FSTTCS 2009, Dagstuhl, Germany. LIPIcs, vol. 4, pp. 287–298 (2009)
13. Razgon, I.: Faster computation of maximum independent set and parameterized vertex cover for graphs with maximum degree 3. J. of Discrete Algorithms 7(2), 191–212 (2009)

14. Robson, J.: Algorithms for maximum independent sets. J. of Algorithms 7(3), 425–440 (1986)
15. Robson, J.: Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, LaBRI, Univsersite Bordeaux I (2001)
16. Tarjan, R., Trojanowski, A.: Finding a maximum independent set. SIAM J. on Computing 6(3), 537–546 (1977)
17. West, D.: Introduction to Graph Theory. Prentice Hall, Englewood Cliffs (1996)
18. Xiao, M., Chen, J.E., Han, X.L.: Improvement on vertex cover and independent set problems for low-degree graphs. Chinese J. of Computers 28(2), 153–160 (2005)
19. Xiao, M.: A simple and fast algorithm for maximum independent set in 3-degree graphs. In: Rahman, M. S., Fujita, S. (eds.) WALCOM 2010. LNCS, vol. 5942, pp. 281–292. Springer, Heidelberg (2010)

Approximation Algorithms for Minimum Energy Multicast Routing with Reception Cost in Wireless Sensor Networks

Deying Li[1], Zewen Liu[2], Yi Hong[1], and Wenping Chen[1]

[1] School of Information, Renmin University of China, Beijing100872, P.R. China
[2] Nanchang University, Nanchang 330031, P.R. China

Abstract. In this paper, we study the minimum energy multicast problem with reception cost in wireless sensor networks. Suppose there are n nodes in the network. Each node v has $l(v)$ transmission power levels to be chosen and its reception cost is $B(v)$ if it receives a message. The problem of our concerning is: given a multicast request, how to find a multicast tree such that the total energy cost of the multicast tree including transmitting cost and reception cost is minimized. We firstly propose a general algorithm Fix-MEM-R-G for the case while each node has the fixed power level. Based on Fix-MEM-R-G algorithm, we propose an approximation algorithm. We also propose a heuristic algorithm for this special case. For the general case that each node has multiple power levels, we propose a general algorithm NF-MEM-R-G and an approximation algorithm based on NF-MEM-R-G algorithm. We also propose a heuristic algorithm for general case.

Keywords: Energy efficient, Multicast routing, Reception cost, Approximation algorithm, Wireless sensor networks.

1 Introduction

Wireless sensor networks have received significant attention in recent years due to their potential applications in battlefield, emergency disaster relief and etc. Broadcast and multicast are important functions in these applications, such as cooperative operation, data dissemination, routing discover, and so on. Energy-efficiency is an important issue in sensor networks, where nodes are powered by batteries that may not be possible to be recharged or replaced during a mission. Therefore, the energy efficient multicast(broadcast) routing is one of the most fundamental and important problems in wireless sensor networks.

There has been a lot of works on energy efficient broadcast/multicast routing in ad hoc networks [1]-[16]. Most of the existing works assume the reception of signals costs no extra energy. However, some research works such as [17] show that power spent in transmit, receive and idle states is between 0.34W and 0.7W. It is necessary to consider reception cost in multicast routing problem. In this paper, we focus on the multicast routing problem with reception cost. We assume that each node v has $l(v)$ transmission power levels and reception cost

$B(v)$ if it receive a signal, and we aim at, for each multicast request, finding a multicast tree that has the minimum energy consumption including transmitting cost and reception cost, we call it as Minimum Energy Multicast with reception cost (MEM-R) problem. We firstly propose an algorithm, called Fix-MEM-R-G for the case where each node has fixed power level. Based on Fix-MEM-R-G algorithm, we have an approximation algorithm for the fix power level case. Moreover, we propose a heuristic algorithm. For the general case where each node has several fixed power levels, we also propose a general algorithm, namely NF-MEM-R-G and an approximation algorithm. Furthermore, a heuristic algorithm is presented for the general case.

The remainder of this paper is structured as follows: Section 2 surveys the related works. Section 3 introduces network model. The algorithms including approximation algorithms solving the MEM-R problem are proposed in Section 4. Finally, we give a conclusion in Section 5.

2 Related Work

There are two types on energy efficient multicast/broadcast tree problem with adjustable transmission range: (1) one is focusing on configuring energy power of each node while each node can adjust its transmission power continuously. That is, given the geometric positions of a set of nodes in a plane, find the transmitting power for each node, such that the energy cost of the multicast/broadcast tree is minimized. (2) Another is focusing on configuring energy power of each node while each node can adjust its transmission power in a discrete way.

For the first type, some energy-efficient broadcast/multicast algorithms were proposed in [2][3], namely BIP (Broadcast Incremental Power), MST (Minimum Spanning Tree), SPT (Shortest Path Tree), and MIP (Multicast Incremental Power). The authors in [4] quantitatively analyzed performances of these three greedy heuristics. In [5], the problem of broadcasting in large ad hoc wireless networks was discussed and a method MLE(Minimum Longest Edge) based on MST was proposed. This algorithm provided a scheme to balance the energy consumption among all nodes. In [6], the GPLE(Greedy Perimeter Broadcast Efficiency) algorithm was proposed. Some fundamental issues associated with energy-efficient multicast were discussed in [7], and several multicast schemes were proposed and evaluated. In [8], the authors first proved the approximation ratios of the pruned based multicast tree algorithms p-SPT, p-MST and p-BIP, and all the three heuristics have $\Omega(n)$ lower bounds. Then two constant approximation ratio algorithms, SPF (Shortest Path First) and MIPF (Minimum Increment Path First), were proposed.

Other works [9]-[16] applied graph theories and techniques to construct broadcast or multicast trees. In [9], the minimum-energy broadcast problem was proved to be NP-hard in general, and an $O(n^{k+2})$ algorithm was proposed for the problem under the assumption that each node is able to reach all the other nodes in the network, where n is the number of nodes and k is the number of transmitters. In [10], authors first gave a formal proof of the NP-hardness for the minimum energy broadcast problem for both geometrical version and graph version. A heuristic algorithm based on MST was proposed, but no performance ratio was given. In [11], another heuristic algorithm for constructing minimum-energy broadcast trees was proposed, which was based on directed Steiner tree. Its performance ratio is n^{ε}, where ε is a constant between 0 and 1. For the special case where each node has the same transmission power level, an algorithm with performance ratio $log^3 n$ was proposed. In [12], an approximation algorithm for the multicast tree problem in symmetric wireless ad hoc networks was proposed, and the solution delivered by the proposed algorithm is within $4lnK$ times of the optimum if the transmission power at each node is finitely adjustable, where K is the number of destination nodes in a multicast request. In [15], we study the multicast tree problem with discrete power levels. [16] presented an energy efficient multicast routing protocol for MANET with minimum control overhead, which created shared multicast tree using the physical location of the nodes for the multicast sessions. In [13], three heuristic algorithms were proposed for broadcast routing in asymmetric wireless ad hoc networks where each node has fixed power, and one of them has an approximation ratio $1 + 2ln(n - 1)$, where n is a number of nodes in the network. In [14], three heuristic algorithms were proposed for multicast in asymmetric wireless ad hoc networks where each node has fixed power.

The above works ignore the energy consumption of receiving signals. However, the authors [17] presented that the energy consumption for receiving is considerable even though it is smaller than that of transmitting. In [18], a polynomial-time near-optimal algorithm was proposed for solving maximum lifetime data gathering problem for sensor networks, in which the reception cost is a constant and irrelevant with distance and depends on the hardware. [19] proposed an energy efficient routing protocol E-PULRP (Energy optimized Path Unaware Layered Routing Protocol) for underwater sensor networks, which considered each node had uniform energy consumption for receiving/processing. In [20], the authors proved that maximizing multicast lifetime with transmitter-receiver power tradeoff is NP-Hard.

In this paper, we study minimum energy multicast tree problem that each node has discrete power levels, while reception cost is considered, which is different from [18]-[20].

3 Network Model and Problem Specification

Suppose there is a set V of n nodes which are deployed in a plane, each node v_i has $l(i)$ transmission power levels, $p_i^1, p_i^2, ..., p_i^{l(i)}$, and its reception cost is $B(v_i)$ if it receives a signal, the reception cost is a constant and irrelevant with distance and depends on the hardware [18].

For a given power assignment, the network is modeled by a directed graph $G = (V, A)$, where V represents the set of n nodes and A the set of arcs in the network. Each node, $v \in V$, $p(v)$ is the transmission power assigned. For any two nodes v_1 and v_2, if v_2 is in the transmission range of v_1 (i.e., $d^{\alpha}(v_1, v_2) \leq p(v_1)$),

α is a constant value between 2 and 4), then there is an arc (a directed link) $(v_1, v_2) \in A$ from v_1 to v_2.

Given a multicast request (s, D), where s is the source and D is a set of destinations, let T be a multicast tree rooted at s for a given power assignment. There are three kinds of nodes in T: the source node that only transmits messages, the nodes that only receive multicast messages, i.e., the leaf-nodes in T, and the nodes that need to transmit/relay multicast messages and receive multicast messages simultaneously, i.e., non-leaf nodes. Let $NL(T)$ denote the set of non-leaf nodes of T which includes the source node. The total energy cost $C(T)$ of T can be represented as:

$$C(T) = \sum_{v \in NL(T)} p(v) + \sum_{v \in T-s} B(v) \tag{1}$$

Our problem is how to, given a multicast request (s, D) and $l(v)$ power levels of transmission $p_v^1, p_v^2, ..., p_v^{l(v)}$ and reception cost $B(v)$ for each node v, find a power assignment such that there exists a multicast tree rooted at s and spanning all nodes in D and total energy cost defined in (1) is minimized. We call it *Minimum Energy Multicast* with reception cost (MEM-R) problem.

4 Algorithms for the MEM-R Problem

The MEM-R problem becomes the G-MEB problem [13] when setting $B(v) = 0$, $\forall v \in V$ and $D = V - \{s\}$. Since the G-MEB problem is proved to be NP-hard [13], therefore the MEM-R problem is NP-hard.

4.1 Fixed Power Level

In this subsection, we study the MEM-R problem for the special case that each node has a fixed power level. The problem is: Given n nodes, while each node v has a fixed power level $p(v)$, and a reception cost $B(v)$ if it receives a message, and given a multicast request (s, D), find a multicast tree such that the total cost is minimized. We call it Fix-MEM-R problem.

The network can be modeled by a directed graph $G = (V, E)$, where V is a set of n nodes and E is a set of directed edges. For any two nodes $u, v \in V$, $(u, v) \in E$ if and only if $d^\alpha(u, v) \leq p(u)$. There are two kinds of weight for each node v. $p(v)$ is the power level of v, i.e., transmission cost, $B(v)$ is the reception cost of v if it receives a message.

Approximation Algorithm for the Fix-MEM-R Problem. In order to design an approximation algorithm for the Fix-MEM-R problem, we first construct an auxiliary node-weighted directed graph $G_w = (V_w, E_w, w)$ based on original directed graph $G = (V, E)$.

For each node $v \in V$, there are two vertices $v^1, v^2 \in V_w$ corresponding to v, and there is a directed edge from v^1 to v^2.

For any directed edge $(u, v) \in E$, there is a corresponding directed edge $(u^2, v^1) \in E_w$. Therefore,

$$V_w = \{v^1, v^2 | \forall v \in V\};$$
$$E_w = \{(v^1, v^2) | \forall v \in V\} \cup \{(u^2, v^1) | \forall e = (u, v) \in E\}.$$

Fig. 1 shows process of constructing auxiliary graph. Fig. 1(a) shows a directed graph of the network model. Fig. 1(b) shows the auxiliary graph corresponding to Fig. 1(a).

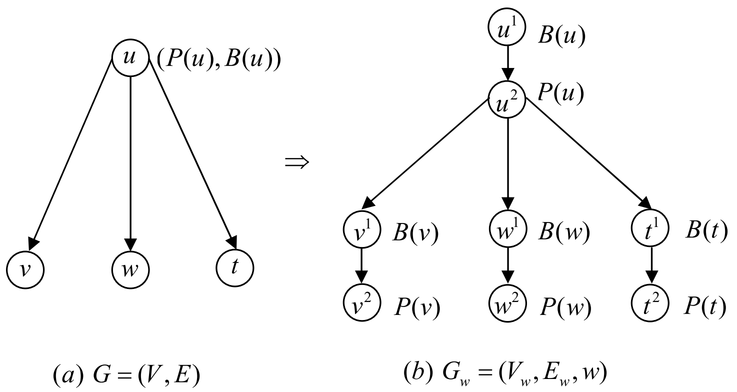


Fig. 1. Constructing an auxiliary graph

We assign a weight to each node in G_w: $w(v^1) = B(v)$ and $w(v^2) = p(v)$ for $v \in V$. For a multicast request (s, D) in G, there is a corresponding multicast request (s^2, D^2) in G_w, where $D^2 = \{v^2 | \forall v \in D\}$.

The Fix-MEM-R problem is transformed to the minimum node-weighted Steiner tree problem on G_w: finding a multicast tree T_w on G_w rooted at s^2 and spanning all nodes in D^2 such that $w(T_w) = \sum_{v \in NL(T_w)} w(v)$ is minimized.

It is easy to get the following theorem:

Theorem 1. *The Fix-MEM-R problem is equivalent to the minimum node-weight directed Steiner tree in G_w. That is, for any multicast request (s, D) in Fix-MEM-R problem, (1) for any multicast tree T in Fix-MEM-R problem, there is a multicast tree T_w in G_w corresponding to T, and vice versa. (2) T_E^{opt} is a minimum energy multicast tree in V rooted at the source s and spanning all nodes in D for Fix-MEM-R problem, T_{NW}^{opt} is a minimum node-weight directed Steiner tree for (s, D) in corresponding graph G_w, then $C(T_E^{opt}) = C(T_{NW}^{opt})$.*

From Theorem 1, we can design a general algorithm for Fix-MEM-R problem as follows:

From the Fix-MEM-R-G algorithm for the Fix-MEM-R problem and Theorem 1, we can get the following theorem.

Algorithm 1. General Algorithm for the Fix-MEM-R Problem(Fix-MEM-R-G Algorithm)

```
Input: $n$ nodes in $V$ while each node has a fixed power level, its reception cost and a multicast request $(s, D)$.
output: a multicast tree rooted at $s$ and spanning all nodes in $D$.
1: Construct a node-weighted auxiliary, directed graph $G_w$;
2: Find an approximate, minimum node-weighted directed Steiner Tree $T_w$ in $G_w$ , such as using algorithms in [14];
3: Transfer $T_w$ to $T$ such that $T$ is a multicast tree for the Fix-MEM-R problem.
```

Theorem 2. *If there is an approximation algorithm for the minimum node-weighted directed Steiner tree problem with ratio θ, then using the algorithm in general algorithm for Fix-MEM-R problem, we can get an approximation algorithm with ratio θ for the Fix-MEM-R problem.*

We use our algorithm: Steiner Tree Based Algorithm in [14] on G_w to get corresponding algorithm for the Fix-MEM-R problem: F-DSTR. From theorem 2 and the theorem in [14], we have:

Corollary 1. *The F-DSTR algorithm has an approximation ratio with $i(i-1)|D|^{1/i}$ and time complexity of $O((2|V|)^i|D|^{2i})$ for any fixed $i > 1$.*

A Heuristic for the Fix-MEM-R Problem. In this subsection, we propose another algorithm with lower time complexity using the original network graph directly for Fix-MEM-R problem since the F-DSTR algorithm has a high time complexity. According to idea of selecting node to join tree [14] and property of the Fix-MEM-R problem, we use a greedy function as following:

Suppose (s, D) is a multicast request and $G = (V, E)$ is the network model for the Fix-MEM-R problem. Let C be a set of nodes which transmit, and U be an un-covered set, i.e., a subset of nodes of D which is not covered by C. Let S be a candidate-set which is candidate for C. $N^+(v)$ denotes a set of out-neighbors.

$$f(v) = \frac{p(v) + \sum\limits_{u \in U \cap N^+(v)} B(u)}{|U \cap N^+(v)|}$$

For a path $P = v_1 v_2 ... v_t$, the total cost of transmission cost and reception cost is

$$w(P) = \sum_{i=1}^{t-1} p(v_i) + \sum_{i=2}^{t} B(v_i)$$

F-NJTR algorithm is represented as follows:

Theorem 3. *Given a request (s, D), the F-NJTR algorithm can output a multicast tree in time $O(|V|^3)$.*

Algorithm 2. F-NJTR Algorithm

Input: V in which each node v has a fixed power level $p(v)$ and a reception cost $B(v)$, and a multicast request (s, D)
Output: A multicast tree T for (s, D).

```
1: $C \leftarrow \{s\}$;
2: $U \leftarrow D - N^+(s)$;
3: $S \leftarrow N^+(s)$;
4:
5: while $(U \neq \emptyset)$ do
6:    if $\exists v \in S$ such that $U \cap N^+(v) \neq \emptyset$ then
7:       Choose $v_i$ such that $f(v_i) = \min\{f(v_k) | v_k \in S$ and $U \cap N^+(v_k) \neq \emptyset\}$
8:       $C \leftarrow C \cup \{v_i\}$;
9:       $U \leftarrow U - N^+(v_i)$;
10:      $S \leftarrow S \cup N^+(v_i) - C$;
11:   end if
12:   Find a shortest path $P = sv_1v_2...v_tu$ from $s$ to node $u$ in $U$;
13:   $C \leftarrow C \cup (P - \{u\})$;
14:   $U \leftarrow U - N^+(v_i)$;
15:   $S \leftarrow S \cup (\cup_{i=1}^{t} N^+(v_i)) - C$;
16: end while
17: Transfer $C$ in $G$ to a multicast tree for $(s, D)$.
```

Proof. It is easy to know that the greedy algorithm can output a multicast tree. In the while-loop, there is at most $|V|$ loops and for each of them, or finding the maximum value takes $O(|V|)$, and finding a shortest path takes $O(|V|^2)$, thus the while-loop can finish in the time of $O(|V|^3)$. In addition, the transformation from C to a multicast tree in the last line takes the time of $O(|V|^2)$. Therefore, the whole algorithm ends in the time of $O(|V|^3)$.

4.2 $l(v)$ Power Levels

In this subsection, we study the MEM-R problem for the general case that each node v has $l(v)$ power levels, we call it as NF-MEM-R problem. We will propose two algorithms for the problem.

General Algorithm for the NF-MEM-R Problem. Given n nodes set V in a plane, each node v_i has $l(i)$ power levels, $p_i^1 < p_i^2 < ... < p_i^{l(i)}$, and $B(v_i)$ is the reception cost of v_i if it receives a signal. We first construct a node-weighted auxiliary directed graph $G = (V, E)$ as follows:

For each node $v_i \in V$, we construct a component $G_i = (V_i, E_i)$ corresponding to v_i.

$$V_i = \{v_i^1, v_i^2, p_{i1}, p_{i2}, ..., p_{il(i)}\}$$

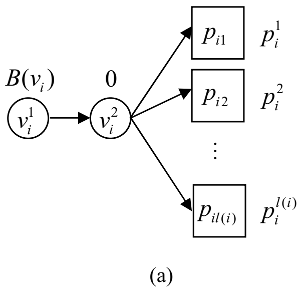
$$E_i = \{(v_i^1, v_i^2)\} \cup \{(v_i^2, p_{ij}) | 1 \leq j \leq l(i)\}$$

Where vertices v_i^1, v_i^2 correspond original node v_i in V, vertex p_{ij} represents original node v_i's transmission power level j, a directed edge (v_i^2, p_{ij}) from v_i^2 to

p_{ij} represents v_i works at its power level j, $1 \le j \le l(i)$. We assign weights to vertices: $w(v_i^1) = B(v_i)$, $w(v_i^2) = 0$, and $w(p_{ij}) = p_i^j, 1 \le j \le l(i)$.

Having a component G_i for every node v_i in V, we construct $G_{aux} = (V_{aux}, E_{aux})$. Let $V_{aux} = \cup_{i=1}^n V_i$ and $E_{aux} = \cup_{i=1}^n E_i \cup E_{diff}$, where E_{diff} is defined as follows: Given two nodes v_i and v_j in V with $i \ne j$, there is a directed edge (p_{ik}, v_j^1) in E_{diff} if and only if v_j is within v_i's the k^{th} transmission range, $1 \le i, j \le n$ and $1 \le k \le l(i)$.

Fig. 2 shows an example that how to construct the auxiliary graph. Fig.2(a) shows a component for node v_i, Fig. 2(b) shows that how to connect components to get the auxiliary graph.



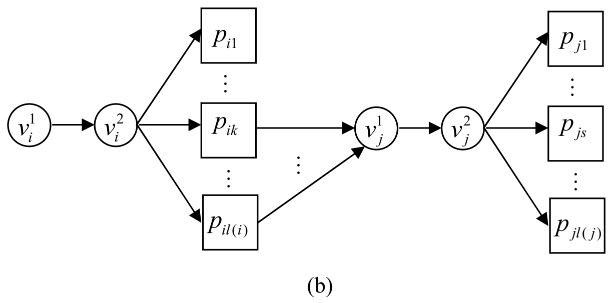


Fig. 2. Example for constructing an auxiliary graph G_{aux}

Having the auxiliary directed graph G_{aux}, let $D^2 = \{v_j^2 | v_j \in D\}$, we denote (s^2, D^2) as a multicast request in G_{aux} corresponding to the multicast request (s, D) in V.

The MEM-R problem can be transformed to the minimum node-weighted Steiner tree problem on G_{aux}: to find a multicast tree T_{aux} on G_{aux} rooted at s^2 and spanning all nodes in D^2 such that $w(T_{aux}) = \sum_{v \in NL(T_{aux})} w(v)$ is minimized. We have the following theorem:

Theorem 4. *The MEM-R problem is equivalent to the minimum node-weight directed Steiner tree problem in G_{aux}. That is: given any multicast request (s, D) in MEM-R problem, we have: (1) for any multicast tree T for the MEM-R problem, there is a multicast tree T_{aux} for the minimum node-weighted Steiner tree* *problem for (s^2, D^2) in G_{aux} corresponding to T, and vice versa. (2) T_E^{opt} is a minimum energy multicast tree in V rooted at the source s and spanning all nodes in D for the MEM-R problem, T_{NW}^{opt} is an optimal tree for (s^2, D^2) in corresponding graph G_{aux} for the minimum node-weighted Stenier tree problem, then $C(T_E^{opt}) = C(T_{NW}^{opt})$.*

Proof. It is obvious that for any multicast tree T of the MEM-R problem, there is a multicast tree T_{aux} for the minimum node-weighted Steiner tree problem for (s^2, D^2) in G_{aux} corresponding to T.

For any multicast tree T_{aux} of the minimum node-weighted Steiner tree problem for (s^2, D^2) in G_{aux}, If T_{aux} meets Fact: (1) no more than one directed edge derived from an original node is included in T_{aux}, then we can set power level for each original node using the information provided by T_{aux} to get the corresponding tree of the MEM-R problem. If T_{aux} does not meet Fact (1), that is, there is one vertex v_i^2 in V_{aux}, there are directed edges $(v_i^2, p_{ij}), (v_i^2, p_{i(j+1)}), ..., (v_i^2, p_{il})$ in T_{aux}, and $l - j \geq 1$. Because the out-neighbors of $p_{ij}, p_{i(j+1)}, , p_{i(l-1)}$ in T must be out-neighbors of p_{il} in G_{aux}, we can modify it to get T_{aux}^1 along modification rule in Fig. 3 such that T_{aux}^1 meets the Fact (1). Therefore, there is a corresponding multicast tree of the MEM-R problem.

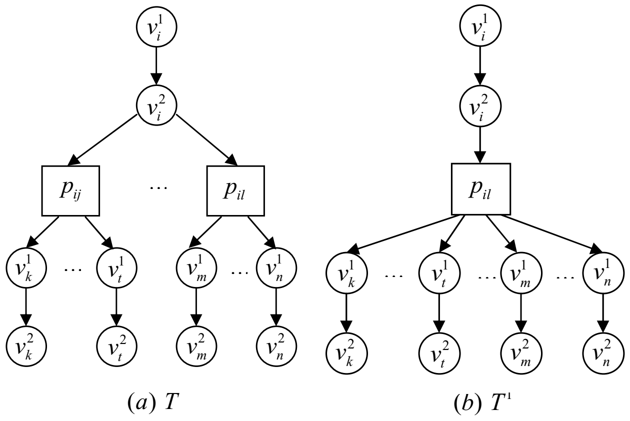


Fig. 3. Modification Rule

In the following, we will prove (2).

We first prove that $C(T_{NW}^{opt}) \leq C(T_E^{opt})$. Since T_E^{opt} is a minimum-energy multicast tree in V rooted at the source s and spanning all nodes in D for the MEM-R problem, according to the construction of G_{aux}, there is a corresponding multicast tree T_{aux} in G_{aux} rooted at s^2 and spanning all vertices in D^2, and the weighted sum of vertices in T_{aux} is equal to the sum of transmission power at the non-leaf nodes and reception cost of all nodes except source s in T_E^{opt}. Since T_{NW}^{opt} is a minimum multicast tree in G_{aux} for (s^2, D^2), then $C(T_{NW}^{opt}) \leq C(T) = C(T_E^{opt})$.

Secondly, we prove that $C(T_{NW}^{opt}) \geq C(T_E^{opt})$.

Since T_{NW}^{opt} is a minimum node-weight directed Steiner tree for (s^2, D^2) in G_{aux}, T_{NW}^{opt} must meet the fact: no more than one directed edge derived from an original node is included in T_{NW}^{opt}. If T_{NW}^{opt} does not meet fact, that is, there is one vertex v_i^2 in V_{aux}, there are directed edges $(v_i^2, p_{ij}), (v_i^2, p_{ij+1}), , (v_i^2, p_{il})$ in T_{NW}^{opt}, and $l - j \geq 1$. We can modify it to get T' such that T' meets the fact, but $C(T_{NW}^{opt}) > C(T')$, which is contradict to that T_{NW}^{opt} is a minimum node-weight directed Steiner tree for (s^2, D^2) in G_{aux}.

Therefore, we can get a multicast tree T in V for the MEM-R problem corresponding to T_{NW}^{opt}. Since T_E^{opt} is a minimum energy multicast tree for the MEM-R problem, then $C(T_E^{opt}) \leq C(T) = C(T_{NW}^{opt})$.

From above discussions, the theorem holds. □

When we get a node-weight directed Steiner tree in G_{aux} which meets the fact that no more than one directed edge derived from an original node is included in T_{aux}, then we can set power level for each original node using the information provided by T_{aux}. This can be done as follows: for a node v_i in V, if there is a directed edge (v_i^2, p_{ij}) in T_{aux}, then set that the power level at v_i is p_{ij}. Then, we get a multicast tree rooted at s and spanning all nodes in D for the MEM-R problem.

From Theorem 3, we can design a general algorithm for the MEM-R problem.

Algorithm 3. General Algorithm for the MEM-R Problem(NF-MEM-R-GAlgorithm)

Input: n nodes in V for each node has its power levels, its reception cost and a multicast request (s, D).
output: a multicast tree rooted at s and spanning all nodes in D.

```
1: Construct a node-weighted auxiliary, directed graph $G_{aux} = (V_{aux}, E_{aux})$;
2: Find a multicast tree $T_{aux}$ in $G_{aux}$ for the minimum node-weighted Steiner tree
   problem, for example, using algorithms in [14].
3: Modify $T_{aux}$ to $T$ such that $T$ meets Fact 1.
4: Set the power level for each node, using the information provided by $T$ to get a
   multicast tree for $(s, D)$.
```

From this general algorithm for the MEM-R problem and Theorem 3, we can get the following theorem.

Theorem 5. *If there is an approximation algorithm $\mathcal{B}$ for the minimum node-weighted Steiner tree problem with ratio θ, then using the algorithm $\mathcal{B}$ in general algorithm for the MEM-R problem, we can get an approximation algorithm with ratio θ for the MEM-R problem.*

Proof. Suppose T_{aux} is a multicast tree got by algorithm $\mathcal{B}$ for the minimum node-weighted Steiner tree problem, and T_{NW}^{opt} is its optimal multicast tree. Since algorithm $\mathcal{B}$ for the minimum node-weighted Steiner tree problem has approximation ratio θ, then

$$C(T_{aux}) \leq \theta C(T_{NW}^{opt})$$

We modify T_{aux} to T_{aux}^1 such that T_{aux}^1 meets the fact (1) and get T^1 which is a multicast tree for the MEM-R problem corresponding to T_{aux}^1. Then, $C(T^1) = C(T_{aux}^1) \leq C(T_{aux})$.

Suppose T_E^{opt} is an optimal multicast tree for the MEM-R problem. From theorem 3, $C(T_{NW}^{opt}) = C(T_E^{opt})$, then:

$$C(T^1) = C(T_{aux}^1) \leq C(T_{aux}) \leq \theta C(T_{NW}^{opt}) = \theta C(T_E^{opt})$$

Therefore, T^1 is θ-approximation solution for the MEM-R problem. The theorem holds. □

We use our algorithm: Steiner Tree Based Algorithm in [14] on G_{aux} to get corresponding algorithms for the NF-MEM-R problem: NF-DSTR. From theorem 4 and the theorem in [14], we have:

Corollary 2. *The NF-DSTR algorithm has an approximation ratio with $i(i-1)|D|^{1/i}$ and its time complexity is $O((2|V|)^i|D|^{2i})$ for any fixed $i > 1$.*

Heuristic for the MEM-R Problem. In this subsection, we will propose another heuristic NF-MIPR for the MEM-R problem, which is similar with MIP. Main idea of heuristic NF-MIPR is as follows. First, we define T only contains node s. Suppose P_{ij} is the minimum power level in which node i can reach node j and P_i is the power level of node i at present (if node i is currently a leaf node, $P_i = 0$). Second, find node j in $V - T$ with minimum incremental cost $P_{ij}^1 = P_{ij} - P_i + B_j$ while B_j being the reception cost of node j and add node j into T, repeat till all the nodes in D are included in the tree.

Algorithm 4. NF-MIPR Algorithm

Input: n nodes in V while each node has its power levels, its reception cost and a multicast request (s, D).

output: a multicast tree rooted at s and spanning all nodes in D.

```
1: $T \leftarrow \{s\}$
2: $\forall i \in V$, $P_i \leftarrow 0$
3: while $((V - T) \cap D \neq \emptyset)$ do
4:     Find a node $i \in T$ and a node $j \in V - T$ such that $P_{ij}^1 = \min\{P_{xy}^1 | \forall x \in T,$
       $\forall y \in V - T$ and $P_{xy}^1 = P_{xy} - P_x + B_y\}$
5:     $T \leftarrow T \cup \{j\}$
6:     $P_i \leftarrow P_{ij}$
7: end while
```

It is easy to get the following corollary.

Corollary 3. *The NF-MIPR algorithm can output a multicast tree in the time of $O(|V|^2)$.*

5 Conclusion

In this paper, we discuss the energy efficient multicast problem with reception cost in wireless ad hoc and sensor networks. We propose two algorithms for special case that each node has a fixed power level. We also get two algorithms for the general case. We prove that one of them is an approximation algorithm respectively.

Acknowledgment. This research was jointly supported in part by the National Natural Science Foundation of China under grant 61070191, the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 10XNJ032, and Research Fund for the Doctoral Program of Higher Education of China(Grant 20100004110001).

References

1. Guo, S., Yang, O.: Energy-aware Multicasting in Wireless Ad Hoc Networks: A Survey and Discussion. Computer Communications 30(9), 2129–2148 (2007)
2. Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: On the Construction of Energy-efficient Broadcast and Multicast Trees in Wireless Networks. In: IEEE INFOCOM 2002, New York (June 2002)
3. Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: Energy-efficient Broadcast and Multicast Trees in Wireless Networks. Mobile Networks and Applications 7, 481–492 (2002)
4. Wan, P.J., Calinescu, G., Li, X.Y., Frieder, O.: Minimum-energy Broadcast Routing in Static Ad Hoc Wireless Networks. In: IEEE INFOCOM 2001, Anchorage, Alaska USA (April 2001)
5. Cheng, M.X., Sun, J., Min, M., Du, D.-Z.: Energy Efficient Broadcast and Multicast Routing in Ad Hoc Wireless Networks. In: Proceedings of 22nd IEEE International Performance, Computing, and Communications Conference, Phoenix, Arizona, USA (2003)
6. Kang, I., Poovendran, R.: A Novel Power-efficient Broadcast Routing Algorithm Exploiting Broadcast Efficiency. In: Proceedings of IEEE Vehicular Technology Conference (VTC), Orlando, pp. 2926–2930 (October 2003)
7. Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: Algorithm for Energy-efficient Multicasting in Static Ad Hoc Wireless Networks. Mobile Networks and Applications 6, 251–263 (2001)
8. Wan, P.J., Calinescu, G., Yi, C.: Minimum-power Multicast Routing in Static Ad Hoc Wireless Networks. IEEE/ACM Transactions on Networking 12(3), 507–514 (2004)
9. Egecioglu, O., Gonzalez, T.F.: Minimum-energy Broadcast in Simple Graphs with Limited Node Power. In: Proceedings of IASED International Conference on Parallel and Distributed Computing and Systems, Anaheim, CA, pp. 334–338 (August 2001)
10. Cagalj, M., Hubaux, J.P., Enz, C.: Minimum-energy Broadcast in All-wireless Networks: NP-completeness and Distribution Issues. In: Proceedings of 8th Annual International Conference on Mobile Computing and Networking, Atlanta, Georgia (September 2002)

11. Liang, W.: Constructing Minimum-energy Broadcast Trees in Wireless Ad Hoc Networks. In: Proceedings of 3th ACM International Symposium on Mobile Ad Hoc Networking and Computing, Lausanne, Switzerland, pp. 112–122 (June 2002)
12. Liang, W.: Approximate Minimum-energy Multicasting in Wireless Ad Hoc Networks. IEEE Transactions on Mobile Computing 5(4), 377–387 (2006)
13. Li, D., Jia, X., Liu, H.: Energy Efficient Broadcast Routing in Ad Hoc Wireless Networks. IEEE Transactions On Mobile Computing 3(2), 144–151 (2004)
14. Li, D., Liu, Q., Hu, X., Jia, X.: Energy Efficient Multicast Tree in Ad Hoc Networks. Computer Communications 30(18), 3746–3756 (2007)
15. Li, D., Zhu, Q.: Approximation Algorithms for Multicast Routing in Ad Hoc Wireless Networks. Journal of Combinatorial Optimization 21(3), 293–305 (2011)
16. Kamboj, P., Sharma, A.K.: Energy Efficient Multicast Routing Protocol for MANET with Minimum Control Overhead (EEMPMO). International Journal of Computer Applications 8(7), 1–11 (2010)
17. Raghunathan, V., Schurgers, C., Park, S., Srivastava, M.B.: Energy-aware Wireless Microsensor Networks. IEEE Signal Processing Magazine 19, 40–50 (2002)
18. Kalpakis, K., Dasgupta, K., Namjoshi, P.: Efficient Algorithms for Maximum Lifetime Data Gathering and Aggregation in Wireless Sensor Networks. In: MobiHoc (2002)
19. Gopi, S., Govindan, K., Chander, D., Desai, U.B., Merchant, S.N.: E-PULRP: Energy Optimized Path Unaware Layered Routing Protocol for Underwater Sensor Networks. IEEE Transaction on Wireless Communications 9(11), 1–6 (2010)
20. Deng, G., Gupta, S.K.S., Varsamopoulos, G.: Maximizing Multicast Lifetime with Transmitter-receiver Power Tradeoff is NP-Hard. IEEE Communications Letters 12(9), 666–668 (2008)

Public Communication Based on Russian Cards Protocol: A Case Study[⋆]

Jia He and Zhenhua Duan[⋆⋆]

Institute of Computing Theory and Technology, and ISN Laboratory
Xidian University, Xi'an 710071, P.R. China
hejia2009@126.com, zhhduan@mail.xidian.edu.cn

Abstract. This paper is concerned with public communication with the Russian Cards protocol. First, a couple of small flaws in [10] are corrected. Then an improved Russian Cards protocol is presented. As a case study, $R(6,31)$(6 players and 31 cards) protocol is used to generate a common password for 5 parties who wish to access a shared file over the Internet.

Keywords: improved Russian Cards protocol, secured communication, password generation.

1 Introduction

Traditional cryptography is composed of symmetrical cryptography and asymmetrical cryptography. In symmetrical cryptography, encryption and decryption use the same key. However, the management of the key is quite difficult and violence analysis could be an effective approach to decrypt the message. The security of asymmetrical cryptography, i.e., public key cryptography, generally depends upon computational problems from number-theory and the assumption that the agents are computationally limited. For instance, Rivest-Shamir-Adleman (RSA)[8] lies on factoring a large product of primes; Digital Signature Algorithm (DSA)[7] relies on the discrete logarithm problem; Elliptic Curve Cryptography (ECC)[2] depends on elliptic curve discrete logarithm problem. All of the above problems are considered to be intractable[9] with computational limits. However, there exist unconditionally secured communication protocols, whose security does not rely on the above restrictions. Some of such protocols have been studied recently in the cryptography and information theory community[3,5]. These protocols show to be safe even against the adversaries with unlimited computational powers, because they guarantee that the adversary cannot learn the secrets for the reasons of information theory rather than computation.

Russian Cards problem was originally presented at the Moscow Mathematics Olympiad in 2000. Several solutions were given to the primary problem with 3 players and 7 cards. Since then some researchers studied the problem[1,4,6,10]. Further, some of them generalized the problem and applied it in public communication. In particular, in

[⋆] This research is supported by the NSFC Grant No. 60910004, 60873018, 91018010, 61003078 and 61003079, 973 Program Grant No.2010CB328102 and SRFDP Grant No.200807010012 and ISN1102001.

[⋆⋆] Corresponding author.

[10], authors put forward a formalized Russian Cards protocol, which can be used in unconditionally secured communication. In this paper, we first correct a couple of small flaws in the algorithm presented in [10] and give an improved protocol. Then as a case study for public unconditional communication, we use the $R(6, 31)$(6 players and 31 cards) protocol to generate a password for 5 parties who wish to access a shared file over the Internet. In addition, the complexity analysis for the public communication is also given in details.

The paper is organized as follows. The Russian Cards Protocol is briefly reviewed in section 2. Further, a few flaws in the protocol are corrected and an improved algorithm is proposed. In section 3, a case of public communication for 5 participants is studied by means of the Russian Cards protocol. Some related works are introduced simply in section 4. Finally, the conclusion is drawn in section 5.

2 An Improved Russian Cards Protocol

In order to describe the improved algorithm clearly and conveniently, we borrow the notations from [10]. We use **R(n, n(n-1)+1)** to represent the Russian Cards problem with n players and $n(n-1)+1$ cards. We call the allocation of cards a **card deal**,the set of cards held by a player a **hand**, and the set of hands appearing in the announcement a **hand set**. Some other notations are also re-used without declaration as seen in Fig.1, and Fig.2. Note that, the notations in column case are different from the original ones presented in [10]. After the allocation of the cards to five parties, there leaves only one card we call **key card**.

$$\begin{array}{r|ccccccc|}
 & & & \text{sharing column}\downarrow & & \text{redundant column}\downarrow & & \\
 & b_{0,0} & \cdots & b_{0,share} & \cdots & b_{0,q} & \cdots & b_{0,n-1} \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\text{redundant row} \longrightarrow & b_{p,0} & \cdots & b_{p,share} & \cdots & b_{p,q} & \cdots & b_{p,n-1} \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\text{hand row} \longrightarrow & b_{row,0} & \cdots & b_{row,share} & \cdots & b_{row,q} & \cdots & b_{row,n-1} \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 & b_{n-1,0} & \cdots & b_{n-1,share} & \cdots & b_{n-1,q} & \cdots & b_{n-1,n-1}
\end{array}$$

Fig. 1. Hand matrix of P_k in row case

The Russian Cards problem has been generalized to n players and $n(n-1)+1$ cards in [10], so that more players are able to communicate using the unconditional protocol. It formalizes picking and deleting rules as well as safe communication condition. Unfortunately, there exist some small flaws in the algorithm given in [10].

2.1 Correction of the Algorithm

Problems in section 3.1.2 of [10]: (1) In the process of generating B^1, sharing card, i.e. $Y[2]$, is placed into the matrix twice, leading to one card left; (2) In the process of

$$\begin{array}{r|ccccccc|}
 & & & \text{hand column } \downarrow & & \text{redundant column } \downarrow & & \\
 & b_{0,0} & \cdots & b_{0,col} & \cdots & b_{0,q} & \cdots & b_{0,n-1} \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\text{redundant row} \longrightarrow & b_{p,0} & \cdots & b_{p,col} & \cdots & b_{p,q} & \cdots & b_{p,n-1} \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\text{sharing row} \longrightarrow & b_{share,0} & \cdots & b_{share,col} & \cdots & b_{share,q} & \cdots & b_{share,n-1} \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 & b_{n-1,0} & \cdots & b_{n-1,col} & \cdots & b_{n-1,q} & \cdots & b_{n-1,n-1}
\end{array}$$

Fig. 2. Hand matrix of P_k in column case

generating B^k, the sharing card is from h_{P_k} instead of $h_{P_{k+1}}$ actually, which is inconsistent with the assumption that *"The first column we call* ***sharing column*** *is filled with a card from $h_{P_{k+1}}$ which we call* ***sharing card****"*. The problems can be illustrated in terms of $R(5, 21)$. Here we assume the card deal is as follows: $h_{P_1} = \{0, 1, 2, 3, 4\}$, $h_{P_2} = \{5, 6, 7, 8, 9\}$, $h_{P_3} = \{10, 11, 12, 13, 14\}$, $h_{P_4} = \{15, 16, 17, 18, 19\}$ and $key\ \ card = \{20\}$. Then we randomly choose card 9 from h_{P_2} as the sharing card of B^1.

According to the algorithm in column case, firstly, we put the sharing card into the first column, the hand cards into the second column, and the key card into the fourth column within the fourth row of B^1, as shown in matrix B^{10}.

Secondly, we divide h_{P_3} into $X[3]$ and $Y[3]$, $X[3] \cup Y[3] = \{10, 11, 12, 13\} \cup \{14\}$, and h_{P_4} into $X[4]$ and $Y[4]$, $X[4] \cup Y[4] = \{15, 16, 17, 18\} \cup \{19\}$. According to step(c)[10], $X[2] = \{5, 6, 7, 8\}$. Thirdly, we put $X[2]$ into B^1 (based on 3.7[10]: $q[i] = \{3, 4, 5, 3\}$) as shown in matrix B^{11}.

After the placement of $X[2]$, we place $X[3]$ and $X[4]$ into B^1, then we get B^{12}.

$$B^{10} = \begin{pmatrix} 9 & 0 & & & \\ 9 & 1 & & & \\ 9 & 2 & & & \\ 9 & 3 & & 20 & \\ 9 & 4 & & & \end{pmatrix}, \quad B^{11} = \begin{pmatrix} 9 & 0 & 5 & & \\ 9 & 1 & & 6 & \\ 9 & 2 & & & 7 \\ 9 & 3 & & 20 & \\ 9 & 4 & 8 & & \end{pmatrix}, \quad B^{12} = \begin{pmatrix} 9 & 0 & 5 & 10 & 15 \\ 9 & 1 & 16 & 6 & 11 \\ 9 & 2 & 12 & 17 & 7 \\ 9 & 3 & & 20 & \\ 9 & 4 & 8 & 13 & 18 \end{pmatrix}$$

Fourthly, we need to put $Y[k]$ into B^1. However, we haven't got $Y[2]$ yet, although, according to step(b) and step(c), we can figure out $Y[2] = b^1_{1,1} = 9$ (sharing card) which has been placed in the first column within the redundant row. Then problem (1) is encountered. Step(d) tells us *"Since there exists only one card in* $Y[2]$*, we place it randomly in any column within the redundant row. We assume the card is in t_ith column, $t_i \in T$, in the redundant row."*, which means $Y[2]$ should be placed in the third or fifth column within the redundant row instead of the first column in this example. As a result, we put $Y[2]$ into B^1 once again. Finally, we get only one empty place in the redundant row to place $Y[3]$ and $Y[4]$. Unfortunately, this causes a conflict.

If we ignored problem (1), and went to step(d), according to Algorithm 3.9[10], we could get B^1; thus, following Algorithm 3.12[10], we could get a wrong B^k, because the sharing card of B^k is from h_{P_k} instead of $h_{P_{k+1}}$; this causes the problem (2). The wrong result is suffered from the statement that *"The first column we call* ***sharing column*** *is filled with a card from $h_{P_{k+1}}$ which we call* ***sharing card****"*.

To deal with the above problems, we modify part of the algorithms given in subsection 3.1.2[10] as follows:

a. In Algorithm 3.9[10], we modify line 5 to **Let** $c \in Y[3]$; **Let** $i \in H$ and line 7 to **for** $k := 4$ to $n - 1$ **do**.

b. In Algorithm 3.12[10], we correct line 5 as **for** $j := 0$ **to** $n - 4$ **do** /*locate the card of $Y[k + 1]$ in redundant row*/ and line 6 as **if** $f_{p,\, t_j} \in Y[k + 1]$ **then**.

2.2 Improvement of the Algorithm

As the row case and the column case are symmetrical, we consider merging the two algorithms of generating B^k into one. Thus, we abandon the algorithm of the column case given in [10], and put forward a symmetrical algorithm for the column case. And then, we merge the two algorithms to cover both row and column cases. In the previous algorithm given in [10], in the row case, the hand row is fixed in the first row and the sharing column is fixed in the first column. Actually, there is no need to make such restrictions. Therefore, we improve the algorithm so that we can randomly choose the hand row and the sharing column. The improvement of the column case is similar to that of the row case and omitted here. Further, the generation process of B^k adhibits some auxiliary matrices. As a result, we can combine the algorithms generating B^k into one. Following the improved algorithm, we can get the matrix B^k as the output according to the input B^1 directly.

Now, we present the improved algorithm in detail. Since the row case and the column case are symmetrical, if some steps cannot be combined to one, we only give the details in column case. For the algorithms given in the following, if some of them only applies to the column case, we will give an explanation, otherwise they can be used in both cases.

Here, we give an example $R(6, 31)$ to show how the improved protocol works in column case. We assume the card deal is as follows: $h_{P_1} = \{30, 17, 6, 22, 25, 11\}$, $h_{P_2} = \{19, 7, 2, 14, 1, 21\}$, $h_{P_3} = \{23, 4, 18, 27, 20, 0\}$, $h_{P_4} = \{8, 10, 9, 5, 12, 13\}$, $h_{P_5} = \{15, 16, 24, 26, 28, 29\}$, *key card* $= 3$. Suppose card 30 as the sharing card of B^1.

Algorithm 2.1: $RC = 1, col = 2, share = 3, p = 4, q = 5$. In column case, $RC = 1$. We select the third column as the hand column. We get the fourth row as the sharing row. Notice that, we cannot place the key card in the sharing row or the hand column(Fig.3).

Algorithm 2.2: $s[0] = 0, s[1] = 1, s[2] = 2, s[3] = 5$; $t[0] = 0, t[1] = 1, t[2] = 3, t[3] = 4$. Then, we get indices $s[4] = \{0, 1, 2, 5\}$, $t[4] = \{0, 1, 3, 4\}$. This algorithm

$$\begin{array}{c|cccccc|}
 & t_0 & t_1 & & t_2 & t_3 & \\
\hline
s_0 & & & 22 & & & \\
s_1 & & & 17 & & & \\
s_2 & & & 6 & & & \\
 & 30 & 30 & 30 & 30 & 30 & 30 \\
 & & & 25 & & & 3 \\
s_3 & & & 11 & & &
\end{array}$$

Fig. 3. Placing h_{P_1} and key card into B^1

Algorithm 2.1. B1RC_Init (Initialization: assignment of row,col,RC,share,p,q)

```
input: integer $r, c, n$; boolean $V\_RC$; array $hand[0 : n-1]$ ;
output: integer $row, col, share, p, q$;
temp variable: set $H$;
Let $row := r$;                         /* hand row: $r \in \{0, 1, \cdots, n-1\}$ */
Let $col := c$;                      /* hand column: $c \in \{0, 1, \cdots, n-1\}$ */
Let $RC := V\_RC$;                     /* RC=0:row case;RC=1:column case */
Let $H := \{0, 1, \cdots, n-1\}$;                    /* range from 0 to n-1 */
Let $share \in H$;              /* RC=0:sharing column; RC=1:sharing row */
if $RC=0$ then                                        /* RC=0:row case */
    Let $p \in H - \{row\}$;                               /* $p \neq row$ */
    Let $q \in H - \{share\}$;                           /* $q \neq share$ */
else
                                                   /* RC=1:column case */
    Let $p \in H - \{share\}$;                           /* $p \neq share$ */
    Let $q \in H - \{col\}$;                               /* $q \neq col$ */
end
```

just works in column case. As for the row case, we can easily get a similar algorithm based on Algorithm 2.2, so we omitted here.

Algorithm 2.3: We randomly divide each hand of parties P_2, P_3, P_4, P_5 into three parts:$X_2 \cup Y_2 \cup Z_2 = \{19, 2, 7, 14\} \cup \{1\} \cup \{21\}$, $X_3 \cup Y_3 \cup Z_3 = \{23, 4, 18, 27\} \cup \{20\} \cup \{0\}$, $X_4 \cup Y_4 \cup Z_4 = \{12, 9, 13, 5\} \cup \{8\} \cup \{10\}$, $X_5 \cup Y_5 \cup Z_5 = \{29, 28, 16, 24\} \cup \{26\} \cup \{15\}$.

We re-use the definition of positive integers od_k and ed_k $(2 \leq k \leq n-1)$ in [10].

Algorithm 2.4: We first put cards of $X_2 = \{19, 2, 7, 14\}$ into B^1. Card 14 is placed as $b^1_{0,4}$, card 2 as $b^1_{1,0}$, card 19 as $b^1_{2,3}$, card 7 as $b^1_{5,1}$. According to the location of X_2, we place cards of $X_3 = \{23, 4, 18, 27\}$, $X_4 = \{12, 9, 13, 5\}$ and $X_5 = \{29, 28, 16, 24\}$ in B^1. For instance, in $s_0^{th}(= 0)$ row, since card 14 from $X_2 = \{19, 2, 7, 14\}$ is in $t_3^{th}(= 4)$ column and $(3 + ed_3) \bmod (6-2) = 2(ed_3 = 6 - 3 = 3)$, we randomly choose card 27 from X_3 and place it in $s_0^{th}(= 0)$ row and $t_2^{th}(= 3)$ column(Fig.4).

$$\begin{array}{c|cccccc|}
 & t_0 & t_1 & & t_2 & t_3 & \\
s_0 & & & 22 & & 14 & \\
s_1 & 2 & & 17 & & & \\
s_2 & & & 6 & 19 & & \\
 & 30 & 30 & 30 & 30 & 30 & 30 \\
 & & & 25 & & & 3 \\
s_3 & & 7 & 11 & & &
\end{array}$$

(a) placing X_2 into B^1

$$\begin{array}{c|cccccc|}
 & t_0 & t_1 & & t_2 & t_3 & \\
s_0 & 5 & 28 & 22 & 27 & 14 & \\
s_1 & 2 & 12 & 17 & 16 & 18 & \\
s_2 & 24 & 23 & 6 & 19 & 9 & \\
 & 30 & 30 & 30 & 30 & 30 & 30 \\
 & & & 25 & & & 3 \\
s_3 & 4 & 7 & 11 & 13 & 29 &
\end{array}$$

(b) placing X_3, X_4, X_5 into B^1

Fig. 4. Placing X_2, X_3, X_4, X_5 into B^1

Algorithm 2.5: We randomly place card of $Y_2 = \{1\}$ in redundant column and s_0^{th} row. According to the location of Y_2, we then place $Y_3 = \{20\}$, $Y_4 = \{8\}$ and $Y_5 = \{26\}$ into B^1. For example, since card 1 is in s_0^{th} row and $0 + ed_3 \bmod (6-2) = 3$, we place card 20 from Y_3 in s_3^{th} row within the redundant column(Fig.5).

Algorithm 2.2: B1C_a(step(a) of generating B^1(column case): S, T)

```
input: integer $p, q, n, col, share$;
output: array $S, T$;
temp variable: integer $i$;
for $i := 0$ to $n - 1$ do                              /* range from 0 to n-1 */
    if $p > share$ then
        if $i < share$ then
            $s_i := i$;
        else if $i = share$ then
            skip;
        else if $i > share$ && $i < p$ then
            $s_{i-1} := i$;
        else if $i = p$ then
            skip;
        else
            $s_{i-2} := i$;
        end
    else
        if $i < p$ then
            $s_i := i$;
        else if $i = p$ then
            skip;
        else if $i > p$ && $i < share$ then
            $s_{i-1} := i$;
        else if $i = share$ then
            skip;
        else
            $s_{i-2} := i$;
        end
    end
    if $q > col$ then
        if $i < col$ then
            $t_i := i$;
        else if $i = col$ then
            skip;
        else if $i > col$ && $i < q$ then
            $t_{i-1} := i$;
        else if $i = q$ then
            skip;
        else
            $t_{i-2} := i$;
        end
    else
        if $i < q$ then
            $t_i := i$;
        else if $i = q$ then
            skip;
        else if $i > q$ && $i < col$ then
            $t_{i-1} := i$;
        else if $i = col$ then
            skip;
        else
            $t_{i-2} := i$;
        end
    end
end
```

$$\begin{array}{c|cccccc} & t_0 & t_1 & & t_2 & t_3 & \\ \hline s_0 & 5 & 28 & 22 & 27 & 14 & 1 \\ s_1 & 2 & 12 & 17 & 16 & 18 & \\ s_2 & 24 & 23 & 6 & 19 & 9 & \\ & 30 & 30 & 30 & 30 & 30 & 30 \\ & & & 25 & & & 3 \\ s_3 & 4 & 7 & 11 & 13 & 29 & \end{array}$$

(a) placing Y_2 into B^1

$$\begin{array}{c|cccccc} & t_0 & t_1 & & t_2 & t_3 & \\ \hline s_0 & 5 & 28 & 22 & 27 & 14 & 1 \\ s_1 & 2 & 12 & 17 & 16 & 18 & 8 \\ s_2 & 24 & 23 & 6 & 19 & 9 & 26 \\ & 30 & 30 & 30 & 30 & 30 & 30 \\ & & & 25 & & & 3 \\ s_3 & 4 & 7 & 11 & 13 & 29 & 20 \end{array}$$

(b) placing Y_3, Y_4, Y_5 into B^1

Fig. 5. Placing Y_2, Y_3, Y_4, Y_5 into B^1

Algorithm 2.3. B1RC_b(step(b) of generating B^1: generating X_k, Y_k, Z_k)

```
input: integer $n$, set $h_{p_k}(k = 2, \dots, n-1)$;
output: set $X_k, Y_k, Z_k (k = 2, \dots, n-1)$;
temp variable: set $M$, integer $k, c$;
for $k := 2$ to $n-1$ do
    $M := h_{P_k}$;                                   /* $M$ is a copy of $h_{P_k}$ */
    for $i := 1$ to $n-2$ do /* pick $n-2$ cards from $M$ and put them into $X_k$ */
        Let $c \in M$;
        $M := M - \{c\}; X_k := X_k \bigcup \{c\}$;
    end
    /* put the remaining two cards from $M$ into $Y_k$ and $Z_k$ respectively
    */
    Let $c \in M$; $M := M - \{c\}; Y_k := \{c\}; Z_k := M$;
end
```

We randomly place card of $Z_2 = \{21\}$ in redundant row and t_3^{th} column. On the basis of the location of Z_2, we then place $Z_3 = \{0\}, Z_4 = \{10\}$ and $Z_5 = \{15\}$ into B^1. For example, since card 21 is in t_3^{th} column and $3 + ed_3 \bmod (6-2) = 2$, we place card 0 from Z_3 in t_2^{th} column within the redundant row(Fig.6).

Finally, we get the announcement matrix of $P_1 : B^1$ (Fig.6(b)). Then we construct B^2 as an example of B^k. According to the Algorithm 2.6, we get B^2 as shown below. Like Algorithm 2.2, Algorithm 2.6 only applies to column case. However, we can get a suitable algorithm for row case based on the same idea. We leave out the details here.

$$B^2 = \begin{pmatrix} 12 & 16 & 2 & 18 & 17 & 8 \\ 23 & 6 & 19 & 9 & 24 & 26 \\ 11 & 13 & 7 & 29 & 4 & 20 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 15 & 0 & 21 & 25 & 10 & 3 \\ 28 & 27 & 14 & 22 & 5 & 30 \end{pmatrix}$$

Algorithm 2.4. B1RC_c(step(c) of generating B^1: put X_k into B^1)

```
1  input: array $S, T$, set $X_k(k = 2, \ldots, n-1)$,integer $n$;
2  output: matrix $B^1_x$;
3  temp variable: set $H$, integer $k, c, d, i$, array $Q$;
4  Let $H := \{0, 1, \cdots, n-3\}$;
5  for $i := 0$ to $n-3$ do                    /* place all cards of $X_2$ in $B^1$ */
6      Let $c \in X_2$; $X_2 := X_2 - \{c\}$;        /* pick a card out of $X_2$ */
7      Let $d \in H$; $H := H - \{d\}$;
8      if $RC=0$ then
9          $b^1_{s_d, t_i} := c$;       /* card $c$ is placed in $s_d{}^{th}$ row and $t_i{}^{th}$column */
10     else
11         $b^1_{s_i, t_d} := c$;       /* card $c$ is placed in $s_i{}^{th}$ row and $t_d{}^{th}$column */
12     end
13     $q_i := d$;
               /* $q_i$ remembers,in $t_i{}^{th}$ column, card $c$ from $X_2$ is in $s_d{}^{th}$ */
14 end
   /* for every other party $P_k$, all cards of $X_k$ are placed in$B^1$ */
15 for $i := 0$ to $n-3$ do
16     for $k := 3$ to $n-1$ do
17         Let $c \in X_k$; $X_k := X_k - \{c\}$;
18         if $n$ is odd then
                       /* card $c$ chosen from $X_k$ is placed in $t_i{}^{th}$ column */
               $d := (q_i + od_k) \bmod (n-2)$;
19         else
20             $d := (q_i + ed_k) \bmod (n-2)$;
21         end
22         if $RC=0$ then
23             $b^1_{s_d, t_i} := c$;
                           /* card $c$ is placed in $s_d{}^{th}$ row and $t_i{}^{th}$ column */
24         else
25             $b^1_{s_i, t_d} := c$;
                           /* card $c$ is placed in $s_i{}^{th}$ row and $t_d{}^{th}$ column */
26         end
27     end
28 end
```

Algorithm 2.5: B1RC_d(step(d) of constructing B^1: put Y_k, Z_k into B^1)

```
1  input: array $S, T$, set $Y_k, Z_k (k = 2, \ldots, n-1)$,integer $n$;
2  output: matrix $B^1$;
3  temp variable: set $H$, integer $k, c, d, i$;
4  Let $H := \{0, 1, \cdots, n-3\}$; Let $c \in Y_2$; Let $i \in H$;
5  if $RC=0$ then
6      $b^1_{p,\, t_i} := c$;
               /* in redundant row,card of $Y_2$ is placed in $t_i{}^{th}$ column */
7  else
8      $b^1_{s_i,\, q} := c$;
              /* in redundant column, card of $Y_2$ is placed in $s_i{}^{th}$ row */
9  end
   /* for every other party $P_k$, card of $Y[k]$ is placed into redundant
   row(column) */
10 for $k := 3$ to $n-1$ do
11     Let $c \in Y_k$;
12     if $n$ is odd then
13         $d := (i + od_k) \bmod (n-2)$;
14     else
15         $d := (i + ed_k) \bmod (n-2)$;
16     end
17     if $RC=0$ then
18         $b^1_{p,\, t_d} := c$;                                /* row case */
19     else
20         $b^1_{s_d,\, q} := c$;                             /* column case */
21     end
22 end
23 Let $H := \{0, 1, \cdots, n-3\}$; Let $c \in Z_2$; Let $i \in H$;
24 if $RC=0$ then
25     $b^1_{s_i,\, q} := c$;
              /* in redundant column, card of $Z_2$ is placed in $s_i{}^{th}$ row */
26 else
27     $b^1_{p,\, t_i} := c$;
              /* in redundant row, card of $Z_2$ is placed in $s_i{}^{th}$ column */
28 end
   /* for every other parity $P_k$, card of $Z_k$ is placed into redundant
   column(row) */
29 for $k := 3$ to $n-1$ do
30     Let $c \in Z_k$;
31     if $n$ is odd then
32         $d := (i + od_k) \bmod (n-2)$;
33     else
34         $d := (i + ed_k) \bmod (n-2)$;
35     end
36     if $RC=0$ then
37         $b^1_{s_d,\, q} := c$;                                /* row case */
38     else
39         $b^1_{p,\, t_d} := c$;                             /* column case */
40     end
41 end
```

Algorithm 2.6. B1toBkC: column case: construct B^k based on B^1

```
input: matrix $B^1$, integer $n, p, k, share, col$;
output: matrix $B^k$;
temp variable: integer $i, j$;
$B^k := B^1$;
for $i := 0$ to $n-3$ do
    $j := (i-(k-1)+n-2)\ mod\ (n-2)$;
    swap($BC^k_{t_j}, BC^k_{t_i}$);          /* swap $t_i{}^{th}$ column with $t_{i-k+1}{}^{th}$ column */
end
for $i := 0$ to $n-3$ do
    $j := (i-(k-1)+n-2)\ mod\ (n-2)$;
    swap($BR^k_{s_j}, BR^k_{s_i}$);          /* swap $s_i{}^{th}$row with $s_{i-k+1}{}^{th}$ row */
end
for $i := 0$ to $n-3$ do          /* locate the card of $Y_k$ in redundant column */
    if $b^k_{s_i,\,q} \in Y_k$ then
        break;
    end
end
$g := b^k_{s_i,\,q}$; $b^k_{s_i,\,q} := b^k_{share,col}$;
for $j := 0$ to $n-1$ do          /* fill the sharing row with card $g$ */
    $b^k_{share,\,j} := g$;
end
/* swap the card of $P_1$ with card of $P_k$ in every other row */
for $i := 0$ to $n-1$ do
    if $i \neq share$ then
        for $j := 0$ to $n-3$ do          /* locate the card of $h_{P_k}$ in $i^{th}$ row */
            if $b^k_{i,t_j} \in h$ then          /* card of $P_k$ is in $i^{th}$ row */
                break;
            end
        end
        $g := b^k_{i,t_j}$; $b^k_{i,t_j} := b^k_{i,col}$; $b^k_{i,col} := g$;
                                        /* swap card $b_{i,t_j}$ with card $b_{i,col}$ */
    end
end
```

$$\begin{array}{c|cccccc|} & t_0 & t_1 & & t_2 & t_3 & \\ s_0 & 5 & 28 & 22 & 27 & 14 & 1 \\ s_1 & 2 & 12 & 17 & 16 & 18 & 8 \\ s_2 & 24 & 23 & 6 & 19 & 9 & 26 \\ & 30 & 30 & 30 & 30 & 30 & 30 \\ & & & 25 & & 21 & 3 \\ s_3 & 4 & 7 & 11 & 13 & 29 & 20 \end{array}$$

(a) placing Z_2 into B^1

$$\begin{array}{c|cccccc|} & t_0 & t_1 & & t_2 & t_3 & \\ s_0 & 5 & 28 & 22 & 27 & 14 & 1 \\ s_1 & 2 & 12 & 17 & 16 & 18 & 8 \\ s_2 & 24 & 23 & 6 & 19 & 9 & 26 \\ & 30 & 30 & 30 & 30 & 30 & 30 \\ & 10 & 15 & 25 & 0 & 21 & 3 \\ s_3 & 4 & 7 & 11 & 13 & 29 & 20 \end{array}$$

(b) placing Z_3, Z_4, Z_5 into B^1

Fig. 6. Placing Z_2, Z_3, Z_4, Z_5 into B^1

3 A Case Study

As a case study, we are concerned with the following public communication.

Five parties want to make a password through consultation by public communications in order to access a shared file over the Internet. It is quite necessary that they should ensure that all of them can get the correct password while others cannot.

Obviously, how to produce the required password is the issue of unconditionally secured communication. Fortunately, we are able to solve this problem by means of the Russian Cards protocol R(6,31). The details of generating the password are as follows.

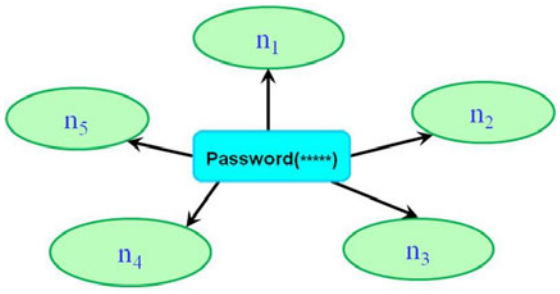


Fig. 7.

As shown in Fig. 7, each party is responsible for generating one digit of the password. At the end of the communication, five digits are obtained in the order of n_1, n_2, n_3, n_4 and n_5 to form the required password.

We use the row case to illustrate how the improved Russian Cards protocol works. To do so, we assume the card deal is as follows: $h_{P_1} = \{30, 17, 6, 22, 25, 11\}$, $h_{P_2} = \{19, 7, 2, 14, 1, 21\}$, $h_{P_3} = \{23, 4, 18, 27, 20, 0\}$, $h_{P_4} = \{8, 10, 9, 5, 12, 13\}$, $h_{P_5} = \{15, 16, 24, 26, 28, 29\}$ and *key card* $= 3$. Suppose that card 30 is the sharing card of B^1, and other parameters are as follows: *row* $= 3$, *share* $= 3$, $p = 4$ and $q = 2$.

According to the improved protocol, a hand matrix is generated for each party as shown below.

$$B^1 = \begin{pmatrix} 20 & 28 & 23 & 30 & 10 & 7 \\ 2 & 4 & 21 & 30 & 24 & 5 \\ 12 & 14 & 8 & 30 & 0 & 29 \\ 22 & 17 & 6 & 30 & 25 & 11 \\ 13 & 15 & 3 & 30 & 27 & 1 \\ 16 & 9 & 26 & 30 & 19 & 18 \end{pmatrix}, \quad B^2 = \begin{pmatrix} 4 & 24 & 6 & 1 & 5 & 22 \\ 17 & 0 & 8 & 1 & 29 & 12 \\ 9 & 25 & 26 & 1 & 18 & 16 \\ 14 & 19 & 21 & 1 & 7 & 2 \\ 15 & 27 & 3 & 1 & 30 & 13 \\ 28 & 10 & 23 & 1 & 11 & 20 \end{pmatrix}$$

$$B^3 = \begin{pmatrix} 25 & 29 & 8 & 27 & 12 & 14 \\ 19 & 11 & 26 & 27 & 16 & 9 \\ 10 & 7 & 6 & 27 & 22 & 28 \\ 0 & 18 & 23 & 27 & 20 & 4 \\ 30 & 1 & 3 & 27 & 13 & 15 \\ 24 & 5 & 21 & 27 & 2 & 17 \end{pmatrix}, \quad B^4 = \begin{pmatrix} 18 & 16 & 26 & 13 & 17 & 19 \\ 7 & 20 & 23 & 13 & 28 & 25 \\ 11 & 2 & 21 & 13 & 4 & 24 \\ 5 & 12 & 8 & 13 & 9 & 10 \\ 1 & 30 & 3 & 13 & 15 & 27 \\ 29 & 22 & 6 & 13 & 14 & 0 \end{pmatrix}$$

In the communication, after P_1, P_2, P_3 and P_4 announce their hand matrices one by one, they know each other's hand. As an exception, P_5 figures out the hands of P_1, P_2, P_3 and P_4 according to his own hand and B^1, B^2, B^3 and B^4 respectively; further, he can also figure out the key card. After that, P_5 announces the key card: $B^5 = 3$. As the key card is known, P_1, P_2, P_3 and P_4 are able to work out P_5's hand. Thus, the communication comes to the end, and every party knows the card deal, i.e., every party's hand. In accordance with each one's hand, part of the password can be generated. See Fig.8.

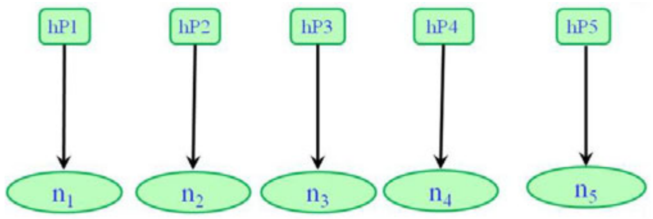


Fig. 8. hPi: hand of party i

In line with one party's hand cards, one digit of the password can be figured out. A variety of rules can be used to get the digit on the basis of the hand cards. Here, we define the rule as $n_i = (c_{i1}^2 + c_{i2}^2 + c_{i3}^2 + c_{i4}^2 + c_{i5}^2 + c_{i6}^2)\ mod\ 36$, in which n_i contributes the i^{th} digit of the password and c_{ij} $(j = 1, 2, 3, 4, 5, 6)$ presents one card of P_i's hand. We choose 36 as the module because passwords generally consist of numbers (0-9) and letters (a-z). If the remainder drops in the range of 0 - 9, n_i equals to the remainder. If the remainder lies in the range of 10-35, we correspond it to a letter from a to z. For instance, remainder 10 indicates $n_i = a$ while remainder 11 means $n_i = b$, etc. The sum of the squares of all the card numbers covers the information of every card in the hand.

As we can see, it is not too complex to figure out the password. Hence, this is a proper rule to generate the password. We take P_2 as an example to explain the generation of the password in detail.

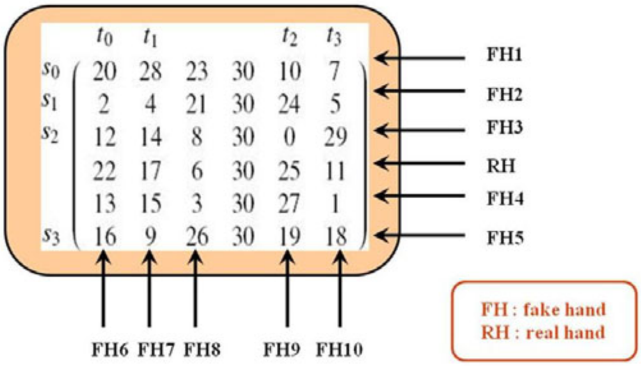
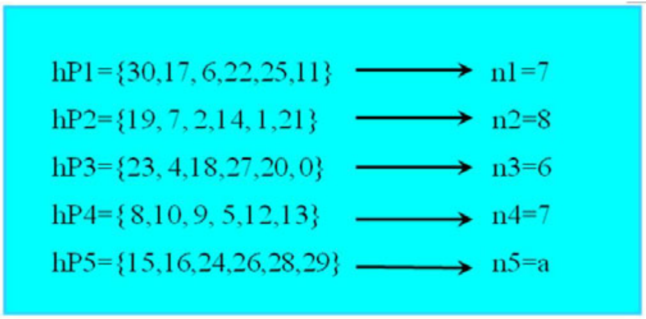


Fig. 9. Hand matrix of P_1

Firstly, according to P_1's hand matrix, P_2 is able to figure out the hand of P_1: 22 17 6 30 25 11, as shown in Fig.9. Then, based on the rule we defined, $n_1 = (22^2 + 17^2 + 6^2 + 30^2 + 25^2 + 11^2)$ *mod* $36 = 7$. Next, P_2 can work out n_2 based on his own hand: $n_2 = (19^2 + 7^2 + 2^2 + 14^2 + 1^2 + 21^2)$ *mod* $36 = 8$. When the communication comes to the end, P_2 knows B^3 and B^4. Moreover, similarly to the process of B^1, P_2 can get n_3 and n_4. On the basis of P_5's announcement, i.e., the key card, P_2 can get the hand of P_5 and n_5. Finally P_2 obtain the password **7867a**, as shown in Fig.10.



hP1={30,17,6,22,25,11} → n1=7
hP2={19,7,2,14,1,21} → n2=8
hP3={23,4,18,27,20,0} → n3=6
hP4={8,10,9,5,12,13} → n4=7
hP5={15,16,24,26,28,29} → n5=a

Fig. 10. Password: 7867a

The participated parties in the generation of the password can get the correct password like P_2, while others can get nothing about it as expected.

Complexity Analysis:
Now, we assume Tinea as an intruder from the Internet. We suppose Tinea knows the protocol used in the communication is the Russian Cards protocol. Since the communication is public, Tinea can get hand matrices B^1, B^2, B^3, B^4 and B^5 easily. We take B^1 as an example to show what Tinea should do to decrypt the message. The announcement of P_1, i.e., the hand set of P_1 is composed of rows and columns except the sharing column of matrix P_1. So, a hand set includes 11 hands (10 fake hands + 1 real hand). As the Russian Cards protocol is open to the public, Tinea can get the information that B^5 is the key card without difficulty. According to the key card, Tinea can remove two hands which insect with the key card from the hand set. As a result, 9 hands are left. However, Tinea could not decide which one is the real hand. For P_2, P_3 and P_4, it can be explained in the same way. As P_5's hand matrix is actually the key card, Tinea has to try $9 \times 9 \times 9 \times 9$ times to get the real card deal. Maybe Tinea can find a card deal that satisfies the conditions well, but she could not make sure whether or not it is the real one. For instance, suppose Tinea tries the following allocation of cards: P_1's hand: 20 28 23 30 10 7, P_2's hand: 4 24 6 1 5 22, P_3's hand: 25 29 8 27 12 14 and P_4's hand: 18 16 26 13 17 19, then Tinea gets P_5's hand: 0 2 9 11 15 21. Although the assignment can satisfy the restriction that the cards should cover all the cards except the key card and every party has a different hand of 6 cards from each other, however, it is not the card deal we assumed. Further, Tinea could not get the correct password at all.

As for the generalized Russian Cards protocol (n players and $n(n-1)+1$ cards), an intruder can filter 3 hands from the matrix according to the key card and $2n-3$ hands are left. Since $n-2$ matrices are generated during the communication, $(2n-3)^{n-2}$ card deal choices will be produced. Therefore, the complexity of decrypting the message comes to be $O(n^n)$. The intruder needs to try all the combinations to achieve the real card deal, otherwise he may get the wrong card deal. What's more, he cannot guarantee the card deal he gets is the real one.

4 Related Works

We use (a, b, c) to denote that the first player holds a cards, the second b cards, and the third c cards.

The Russian Cards problem is generalized to (k, k, l) in [1]. Authors analyze the problem and the generalization (k, k, l) in the framework of Dynamic Epistemic Logic. They draw the conclusion that there is no one-announcement solution to the Russian Cards problem in general and no two announcement solution to (k, k, l) problem. In [6], authors talk about the solutions to the original Russian Cards problem $(3, 3, 1)$, and put forward a two-step protocol. Further, they increase the number of cards while they keep the number of players fixed and explore solutions to the updated problem (a, b, c). Compared with [6], authors of [10] generalize the primary problem to n players and $n(n-1)+1$ cards so that it can be used to deal with multi-party communication instead of only two parties and formalize a Russian Cards protocol. However, the number of cards is relevant to the the number of players in [10]. As a result, neither the protocol given in [10] nor the improved one given in this paper work well in common case with n players and m cards. Also none of them does the two-step protocol presented in [6]. This remains open for further investigation.

In [6], authors study the problem in purely combinatorial terms. In [10], authors also give formalized protocol without practical communication examples. Therefore, we show how the Russian Cards protocol can be used in practical communication and give a paradigm as an application in this paper.

5 Conclusion

We corrected and improved the original Russian Cards protocol algorithm. The improved protocol can generate more complex and random matrices. Therefor, it will be more difficult to decrypt the message. Further, a case study using the protocol for communication is given. However, the paradigm is just the generation of a password. In the future, we will further explore the communication on the text using the Russian Cards protocol.

References

1. Cryriac, A., Murali Krishnan, K.: Lower Bound for the Communication Complexity of the Russian Cards Problem. CoRR abs/0805.1974 (2008)
2. Menezes, A.J.: Elliptic Curve Public Key Cryptosystems. Springer, Heidelberg (1993) ISBN: 9780792393689
3. Fischer, M.J., Wright, R.N.: Bounds on Secret Key Exchange Using a Random Deal of Cards. Journal of Cryptography 9(2), 71–99 (1996)
4. van Ditmarsch, H.: The Russian Cards Problem: a Case Study in Cryptography with Public Announcements. In: Proceedings of AWCL 2002 (Australasian Workshop on Computational Logic) Canberra, pp. 47-67. Technical Report TR2002/6, Department of Computer Science, University of Melbourne (2002)
5. Makarychev, K.: Logicheskie Voprosy Peredachi Informacii (Logical Issues of Information Transmission). Master's thesis, Moscow State University, Diplomnaja rabota, part1 (2001)
6. Albert, M.H., Aldred, R.E.L., Atkinson, M.D., van Ditmarsch, H.P., Handley, C.C.: Safe Communication for Card Players by Combinatorial Designs for Two-step Protocols. Australasian Journal of Combinatorics 33, 33–46 (2005)
7. National Institute of Standards and Technology, A Proposed Federal Information Processing Standard for Digital Signature Standard (DSS), Federal Register, vol. 56(169), pp. 42.980-42.982, (August 30,1991)
8. Rivest, R., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
9. Vasilenko, O.: Number-theoretic Algorithms in Cryptography (Translations of Mathematical Monographs). American Mathematical Society, Boston (2006)
10. Duan, Z., Yang, C.: Unconditional Secure Communication: a Russian Cards Protocol. Journal of Combinatorial Optimization 19(4), 501–530 (2010)

Minimum Latency Data Aggregation in Wireless Sensor Network with Directional Antenna

Hui Liu[1], Zewen Liu[2], Hongwei Du[3], Deying Li[1,⋆], and Xianling Lu[1]

[1] School of Information, Renmin University of China, Beijing 100872, P.R. China
deyingli@ruc.edu.cn
[2] Nanchang University, Nanchang 330031, P.R. China
[3] Department of Computer Science and Technology, HITSGS, China

Abstract. Real-time data aggregation is one of important issues in wireless sensor network. Using directional antenna to transmit data can save energy and reduce interference. In this paper, we study the minimum latency data aggregation problem with directional antenna model under protocol interference model. We propose an approximation directional data aggregation algorithm to schedule data transmissions, which can save energy and reduce interference to reduce latency.

Keywords: data aggregation, conflict-aware, minimum latency, directional antenna, wireless sensor network.

1 Introduction

A wireless sensor networks (WSN) consists of sensor nodes using radio transmissions to organize a temporary network dynamically. It has played an important role in many practical applications such as fire detection, battlefield surveillance and healthcare [1]. In these applications, quite often we need to collect(or aggregate) data from these sensor nodes to a fixed sink. Real-time is very important in most applications. Therefore, reducing the data aggregation latency is very important in wireless sensor networks.

In wireless sensor networks, each sensor node is powered by battery that may not be possible to be recharged or replaced during a mission. Consequently, the limited energy makes the energy efficiency become one of the primary issues in such networks. Compared with the omni-directional antenna, the directional antenna could save energy and reduce the number of received duplicate packets. On the other hand, as the spatial reuse it will produce less interference and decrease the latency of the data aggregation process.

In this paper, we focus on minimum latency conflict-aware aggregation schedule problem under the directional antenna model. As many sensors which lie far away from the sink node need intermediate nodes to relay their messages, it is necessary to schedule the data aggregation. In this paper, we consider the protocol interference model under the directional antenna model. We propose an approximation algorithm for the problem.

⋆ Corresponding author.

The rest of this paper is organized as follows. Section 2 reviews related works. Section 3 describes directional antenna model, network model and problem definition. Section 4 discusses the minimum latency data aggregation problem and proposes an approximation algorithm. Section 5 gives the proof about algorithm performance ratio.

2 Related Work

There have been many works for the minimum latency data aggregation problem. In [2], the authors studied the minimum data aggregation time (MDAT) problem in wireless sensor networks on unit disk graph, and designed a $(\Delta - 1)$-approximation algorithm for the MDAT problem. Huang et al. [3] also focused on the MDAT problem in which the transmission range equals to interference range, and proposed an algorithm with a latency bound of $23R+\Delta-18$, where R is the transmission radius. Authors of [12] proposed a distributed approximation algorithm with latency bound $24D + 6\Delta + 6$, while D is the network diameter. [5] studied the minimum-latency aggregation schedule (MLAS) problem in which the interference radius is ρ times the transmission range. Shang et al. [4] studied the minimal convergecast time problem and P.-J. Wan et al. [6] focused on utilizing the multiple channels to speed up some communications problems. Q. Zhu et al. [13] studied the minimum latency conflict-aware many-to-one data aggregation scheduling problem considering the carrier sensing range and proposed an approximation algorithm with a nearly constant ratio. P.-J. Wan et al. [11] studied the minimum-latency beaconing schedule(MLBS) which seeks a schedule for beaconing with the shortest latency in synchronous multihop wireless networks and presented strip coloring approximation algorithm for MLBS under general protocol interference model.

Recently directional sensors are widely used in many applications. Guo et al. [7] presented a distributed constant-factor approximation algorithm for the maximum multicast lifetime problem under directional communications. Kathiravan et al. [8] proposed a directional broadcast protocol (DBP) to alleviate broadcast storm in ad hoc networks using directional antennas. Li et al. [9] discussed the energy efficient broadcast problem with a directional antenna model which could select propagation area. Yang et al. [10] designed an energy efficient broadcast algorithm with network coding mechanism and directional antenna to reduce the transmission number of the data packet and energy consumption.

[15] first discussed the directional data aggregation problem, and they proposed a directional flooding with data aggregation using hop-count values of sensor node for wireless sensor network. However, it just uses flooding to send data, there are many unnecessary transmissions. In this paper, we will propose a new deterministic algorithm to deal with conflict-aware data aggregation problem with directional antenna for reducing aggregation latency.

3 Network Model and Directional Antenna Model

3.1 Directional Antenna Model

In directional wireless sensor networks, each node device is capable of sensing, processing data, and communicating. All sensors are equipped with directional sending antennas and omni-directional receiving antennas. And the propagating area of the sending antenna is a sector of the disk centered at the sending sensor node with unit transmission radius. It can be effectively controlled by the beam-width θ and the antenna orientation p which can steer to any desired direction, where the orientation p denotes the bisector of included angle of the propagation sector. The interference range of the sending antenna is also a sector of the disk centered at the sending sensor node but with the interference radius r_d, $r_d \geqslant 1$. Fig.1 shows an example of sensor node's directional antenna model.

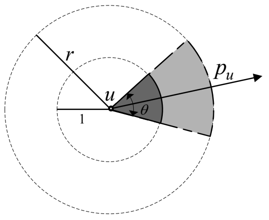


Fig. 1. The propagation area and interference area of node u

3.2 Network Model

There are n sensor nodes and a sink node s which are deployed on 2-D plane. Each sensor node transmits data using directional antenna, while all sensor nodes including the sink node receive data using omni-directional antenna.

With the directional antenna model, node v can receive messages from node u only when v is located within u's propagating area. If v is located within the propagating area of u, there is a directed edge from node u to v in the corresponding communication graph.

While the data transmission is deterministic and proceeds in synchronous rounds controlled by a global clock, the interference may be generated when the nodes simultaneously transmit messages in the network. Now we consider the potential conflicts in parallel transmissions. Two parallel transmissions from u and v can be expected to succeed in the same slot only if they avoid all the following three conditions:

(1) at least one of nodes u and v is within the other's propagation area, as Fig.2 (a)(b) shown;
(2) there is a node w other than node u and node v such that w is within u's propagation area and v's interference area respectively, as Fig.2(c) shown;

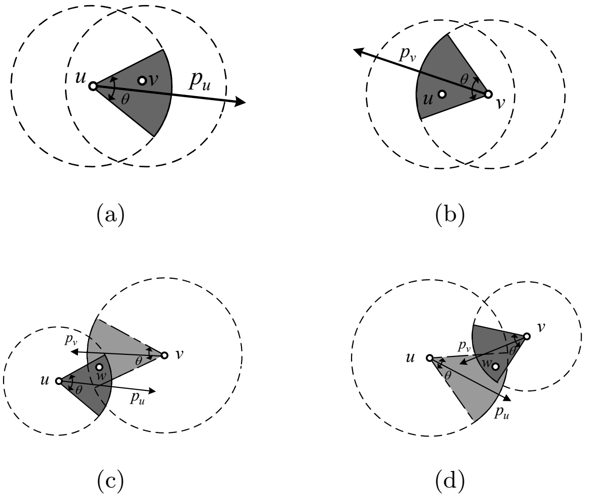


Fig. 2. Three possible scenarios of conflict between node u and node v under the protocol interference model

(3) there is a node w other than node u and node v, which is within v's propagation area and u's interference area respectively, as Fig.2(d) shown.

Obviously, we know that the directional antenna can not only save the energy but also reduce the interference in the process of transmission compared with the omni-directional antenna.

3.3 Problem Definition

Suppose that there is a set of sensors V including sink node s. The parameters of the directional antenna are beam-width θ and the tunable orientation p. The transmission radius is unit and interference radius is r_d, $r_d \geq 1$. Given V, θ, p and r_d for each sensor, the network can be modeled as a directed graph $G = (V, E)$. For any two nodes u, v, there is a directed edge $\overrightarrow{(v, u)}$ if and only if u is in the propagation area of v. We suppose that when each node receives the packets from its children, it can aggregate these packets with itself packet into one packet.

The data aggregation schedule is a sequence of parallel transmissions. We formulate it as $A_{sch} = \{A_1, A_2, \cdots, A_l\}$, where $A_i = \{(v, u) | (v, u)\, is\, in\, the\, time\, slot\, i\}$ $(1 \leq i \leq l)$. Let $S_i = \{v | (v, u) \in A_i\}$ and $R_i = \{u | (v, u) \in A_i\}$ $(\forall 1 \leqslant i \leqslant l)$, where S_i and R_i respectively represent that the senders and the receivers of transmissions scheduled in time slot i. The number l is called the data aggregation latency.

The parallel and collision-free transmissions in a schedule need to satisfy that (1) a node can either send or receive at one time slot. (2) each node can receive data from at most one of its neighbors in one schedule. (3) every node just transmits data once and doesn't receive data after being scheduled to transmit. (4) the data of each node can be sent to the sink node. (5) the transmissions in each time slot are collision-free.

Definition 1. *The minimum latency conflict-aware data aggregation scheduling with directional antenna (LMASD) problem: Given a sink node s that collects the information of all sensors, the parameters unit* 1 *and r_d, and the directional antenna beam-width θ and the tunable orientation p, to find a conflict-aware data aggregation schedule with minimum latency by the directional antenna.*

Theorem 1. *The MLASD problem is NP-hard.*

Proof: The Minimum-Latency Aggregation Schedule (MLAS)[2] is a special case of our MLASD problem. This is because when $r = 1$ and $\theta = 2\pi$ for each sensor, the MLASD problem becomes the MLAS problem. Since the MLAS problem is NP-hard [2], thus the MLASD problem is NP-hard.

4 Algorithm

In order to construct a data aggregation tree and its scheduling, we first construct an undirected connected graph G_o by setting $\theta = 2\pi$, where there is an edge (v, u) if and only if the distance between u and v is not larger than 1. Secondly, we construct a data aggregation tree based on G_o, then initially determine the orientation of directional antenna of each sensor. Finally, adjust the initial data aggregation tree to a final directional data aggregation tree and construct a conflict-aware data aggregation scheduling under the protocol interference model.

4.1 Constructing Initial Data Aggregation Tree

In this subsection, we will structure a directional data aggregation tree. At first, we construct a BFS tree on G_o, and find a MIS. And then, we build an initial directional data aggregation tree and determine the initial orientation of the directional antenna of each transmitting sensor.

Constructing a BFS Tree. Firstly, each sensor node including the sink node s opens its directional propagating antenna with the beam-width 2π to get its neighbor set $NG_o(v)$, the omni-directional connected graph $G_o = (V, E)$ is got, which is an UDG. We construct a BFS tree with the root s denoted by T_{BFS} for G_o.

Secondly, we construct a MIS D of G_o and a partition $D_0, D_1, \cdots, D_L$ of the set D got layer by layer, L is the depth of the BFS tree. The nodes in D denote the dominator nodes. We get the MIS from the root s to the leaves of the BFS tree T_{BFS}. Initially, set $D = \emptyset$, $D_0 = \emptyset$. We firstly construct one subset of the partition of MIS D_1 in the 1^{st} level by checking each node in $L(1)$ and set $D = D \cup D_1$. In the next steps, for each level i, we repeat checking each node in

$L(i)$ whether it is independent of all nodes in D by the graph G_o till $i = L$. If it is, we take it in the MIS D and its subset D_i. The algorithm is similar with our work in [13]. We denote as $BFS(G_o, s)$, where we get each node v's parent node $Fa(v)$ except sink s and output the independent set D and its one partition $D_0, D_1, \cdots, D_L$ and the node set of $L(0), L(1), \cdots, L(L)$ for each level in T_{BFS}.

Constructing Initial Data Aggregation Tree. We construct an initial directional aggregation tree G_d by the MIS D and its partition. The main idea is that we connect all the nodes in D by some nodes which denote the connectors and achieve a directed graph. Initially, G_d contains all nodes of T_{BFS} and no edges. Starting from the 1^{st} level, we choose some nodes to be connectors of the nodes in D in T_{BFS}. Let C_i denote the connector set of nodes in D_i , which is the parent set of nodes in D_i. For $\forall 1 \leq i \leq L$, we need to find a connector u in the $(i-1)^{th}$ level which connects to node v in D_i. Then add u into C_i and its directed edge $\overrightarrow{(v,u)}$ into G_d and initialize the orientation $p_v = \overrightarrow{(v,u)}$. Nextly, we are able to find one dominator w to be its parent for each node u in C_i, where $w \in D_{i-1} \cup D_{i-2}$, $D_{i-1} \subseteq L(i-1)$ and $D_{i-2} \subseteq L(i-2)$, which are easily obtained by the property of MIS. We take $\overrightarrow{(u,w)}$ into G_d and $p_u = \overrightarrow{(u,w)}$. After that, all the dominators can be connected to the sink s. At last, check the directed graph G_d and we may find some nodes isolated in G_d. We set these nodes to be $NC = V - (\bigcup_{i=1}^{L} C_i \cup D)$. They are certain to have dominators in D by the MIS so we do the same procedure as C_i. Consequently, the graph G_d is the initial directional aggregation tree. The algorithm is formally represented as follow:

4.2 Scheduling for the Directional Data Aggregation

In this subsection, we schedule the conflict-aware directional data aggregation by the protocol interference model shown in Fig 2.

Before constructing the conflict-aware directional data aggregation schedule, we get the following lemma.

There are two transmissions "$a \to b$" and "$c \to d$" which are non-conflicting parallel if they can be scheduled in one time slot.

Lemma 1. *Two parallel transmissions "$a \to b$" and "$c \to d$" are said to be non-conflicting if and only if any one of the following three conditions is not satisfied (based the protocol interference model shown in section 3.2):*

(1) If $d(a,c) \leq 1$, then $\widehat{(p_a, \overrightarrow{ac})} \leq \theta/2$ and $\widehat{(p_c, \overrightarrow{ca})} \leq \theta/2$;

(2) If $d(a,d) \leq r$, then $\widehat{(p_a, \overrightarrow{ad})} \leq \theta/2$;

(3) If $d(c,b) \leq r$, then $\widehat{(p_c, \overrightarrow{cb})} \leq \theta/2$.

Construct conflict-aware sub-schedule. In this subsection, we construct the sub-schedule for given sender set S and receiver set B based on given connected graph G_o. The schedule from the senders to their receivers is given as follows. Let A be the set of parallel transmissions shown in Lemma 1, which can be

Algorithm 1. Initial-DA-Tree($G_o, T_{BFS}, \theta, D, \{D_0, D_1, \cdots, D_L\}$)

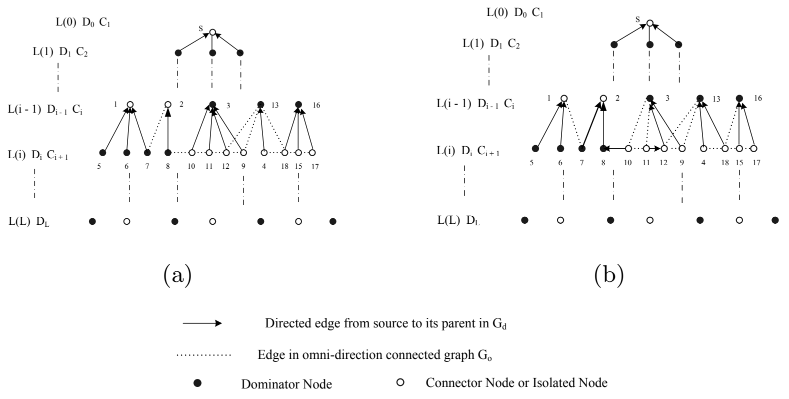
```
Input: A connected sensor network $G_o$, a breadth first search tree $T_{BFS}$ , the beam-
    width $\theta$, a maximal independent set $D$ and its partition $\{D_0, D_1, \cdots, D_L\}$
Output: The initial directional data aggregation tree $G_d$, some connector sets
    $\{C_1, C_2, \cdots, C_L\}$, and an isolated node set $NC$
 1: Let $p_v$ denote the orientation of node $v$'s directional propagation antenna, $\theta_v$ denote
    the beam-width of node $v$'s directional propagation antenna.
 2: $G_d = (V, E'_d)$, where $E'_d = \emptyset$
 3: for each node $v \in V$ do
 4:    $\theta_v = \theta$
 5: end for
 6: $C_1 = \{s\}$
 7: for each node $v \in D_1$ do
 8:    $E'_d = E'_d \cup \{\overrightarrow{(v,s)}\}$, $p_v = \overrightarrow{(v,s)}$
 9: end for
10: for $i = 2$ to $L$ do
11:    $C_i = \emptyset$
12:    for each node $v \in D_i$ do
13:       $u = Fa(v)$, $p_v = \overrightarrow{(v,u)}$, $E'_d = E'_d \cup \{\overrightarrow{(v,u)}\}$, $C_i = C_i \cup \{u\}$
14:       Select one of $u$'s dominators $w$, $w \in D_{i-1} \cup D_{i-2}$
15:       $Fa(u) = w$, $p_v = \overrightarrow{(u,w)}$, $E'_d = E'_d \cup \{\overrightarrow{(u,w)}\}$
16:    end for
17: end for
18: $NC = V - (\bigcup_{i-1}^{L} C_i \cup D)$
19: for each node $v \in NC$ do
20:    Select one of $v$'s dominators $u$, $u \in D$
21:    $Fa(v) = u$, $E'_d = E'_d \cup \{\overrightarrow{(v,u)}\}$, $p_v = \overrightarrow{(v,u)}$
22: end for
23: return $G_d$, $\{C_1, C_2, \cdots, C_L\}$, and $NC$
```

scheduled at the same time slot. First, we set $A = \emptyset$, and put a directed edge $\overrightarrow{(v, F_a(v))}(v \in S)$ into A and remove v from set S. Then select another edge $\overrightarrow{(u, Fa(u))}(u \in S)$ to check whether it is parallel with any transmission in A, by the conditions given in Lemma 1. If it won't conflict with any one, add it into A and move u out of set S. Repeat these steps till there is no collision-free edge with transmissions in A. Finally, repeat this process till all nodes in S can be scheduled.

When we find no conflict-free transmission, we can re-adjust some sensors' direction such that more non-conflicted transmissions can be transmitted in same slot. Two cases for senders need to be considered. Set $M = \bigcup_{i=1}^{L} C_i \cup NC = V - D$.

Case 1: $S \subseteq M$. It is either the isolate node NC set or the connector set $C_i, 1 \leq i \leq L$. When $S = C_i$, we pick up a node $v(v \in H)$ (H is the set of nodes in S which are not senders in A), then check each directed edge from v to each node u of v's neighbors which are in $B(B = L(i-1) \cup L(i-2))$, that is $u \in ((NG_o(v) \cap (L(i-1) \cup L(i-2)) - \{Fa(v)\}$, whether this directional transmission $\overrightarrow{(v,u)}$ conflicts with any transmission in A or not. If no, we add the transmission $\overrightarrow{(v,u)}$ into A, delete the directed edge $\overrightarrow{(V,Fa(v))}$ from the edge set E_d of the input data aggregation T_d and add $\overrightarrow{(v,u)}$ into E_d, and change v's parent $Fa(v) = u$ and remove v from H. Let $Tag(v)$ denote whether node v has been scheduled as a sender. Repeat to check every neighbor in B of all nodes of H till there is no suitable transmission. Let $Tag(v)$ denote whether node v has been scheduled as a sender. Here, we don't care whether u is dominator, we only need that value of $Tag(u)$ must be zero, i.e., we only need u didn't send data till now. For $S = NC$, nodes in NC are isolate nodes and will be scheduled at first, and their neighbors may be in all the levels of the BFS tree, that is $B = V$.

Case 2: $S \subseteq D$, to take $S = D_i$ as an example, we pick up a node $v(v \in H$, H is a parent set) and then check each directed edge from v to each node u of v's neighbors which are in set $B(B = L(i-1))$, where $u \in ((NG_o(v) \cap L(i-1)) - \{Fa(v)\}$, whether this directional transmission $\overrightarrow{(v,u)}$ conflict with any transmission in A or not. If no, we add the transmission $\overrightarrow{(v,u)}$ into A, delate the directed edge $\overrightarrow{(V,Fa(v))}$ from E_d and add $\overrightarrow{(v,u)}$ into E_d, set $Tag(v) = 1$ and change v's parent $Fa(v) = u$ and remove v from H. Repeat to check every neighbor in B of all nodes of H till there is no suitable transmission. Because node u in B is in the upper level of node v in S, that the value of $Tag(u)$ is 0 means that receiver u does not been scheduled as a sender.



L(0) D0 C1
L(1) D1 C2
L(i - 1) Di-1 Ci
L(i) Di Ci+1
L(L) DL
s
1
2
3
13
16
5
6
7
8
10
11
12
9
4
18
15
17
(a)
(b)
Directed edge from source to its parent in Gd
Edge in omni-direction connected graph Go
Dominator Node
Connector Node or Isolated Node

Fig. 3. An example of the procedure of adjusting initial directional data aggregation tree in the i^{th} lever

Repeat these steps till all the nodes in S are checked.

Now, we give an example for adjusting initial directional data aggregation tree G_d in the i^{th} lever. Fig.3(a) presents the i^{th} and $(i-1)^{th}$ level of G_d. Here, $C_{i+1} = \{4, 9, 10, 11, 12, 15, 17, 18\}$ and $D_i = \{5, 6, 7, 8\}$. The first step is to schedule nodes in C_{i+1} to their parents. The sender node with smaller label is checked firstly. Checking these transmissions by conditions in lemma 1, we find an initial collision-free schedule $A = \{\overrightarrow{(9,3)}, \overrightarrow{(4,13)}, \overrightarrow{(15,16)}\}$. Then

$H = C_{i+1} - \{4, 9, 15\} = \{10, 11, 12, 17, 18\}$. Now, we find their new father nodes from these nodes' neighbors in G_o in the same level i^{th} and upper level $(i-1)^{th}$ if there are directed edges from nodes in H to their neighbors. For node 10, the directed edge $\overrightarrow{(10,8)}$ does not conflict with any transmission in A and $Tag(8) = 0$, change the parent of 10 to its neighbor 8 and add $\overrightarrow{(10,8)}$ into A and E_d and move $\overrightarrow{(10,3)}$ out of E_d, where $T_d = (V, E_d)$ is the finial directional data transmission tree. For node 11, we can add $\overrightarrow{(11,12)}$ into A. For node 12, we find the directed edges $\overrightarrow{(12,9)}$, $\overrightarrow{(12,11)}$ and $\overrightarrow{(12,13)}$ to conflict with $\overrightarrow{(9,3)}$ or $\overrightarrow{(11,12)}$ in A so that the parent of node 12 can not be changed. Repeat these processes for node 17 and 18 and get the first conflict-aware schedule $\{\overrightarrow{(9,3)}, \overrightarrow{(4,13)}, \overrightarrow{(15,16)}, \overrightarrow{(10,8)}, \overrightarrow{(11,12)}\}$. Moreover, we construct the next schedule from sender set $H = \{12, 17, 18\}$ and find the initial schedule $\{\overrightarrow{(12,3)}, \overrightarrow{(17,16)}\}$. Nextly, for node 18 its neighbor 4 has $Tag(4) = 1$ which means that it has been scheduled, so we check its another neighbor 13 to find 13 is suitable. Then the second conflict-aware is $\{\overrightarrow{(12,3)}, \overrightarrow{(17,16)}, \overrightarrow{(18,13)}\}$. As $H = \emptyset$, the connector set C_{i+1} has been scheduled over and we catch two conflict-aware schedules $\{\{\overrightarrow{(9,3)}, \overrightarrow{(4,13)}, \overrightarrow{(15,16)}, \overrightarrow{(10,8)}, \overrightarrow{(11,12)}\}, \{\overrightarrow{(12,3)}, \overrightarrow{(17,16)}, \overrightarrow{(18,13)}\}\}$, which need two time slots. The second step is to schedule nodes in D_1 to their parents. We find an initial collision-free schedule $A = \{\overrightarrow{(5,1)}, \overrightarrow{(8,2)}\}$. Then, $H = \{6, 7\}$, we adjust the parents of the nodes in H to their neighbors in the upper level $(i-1)^{th}$ and find that there is no suitable neighbor to be as their new parents. Thus, the first schedule is $\{\overrightarrow{(5,1)}, \overrightarrow{(8,2)}\}$. Nextly, we schedule $H = \{6, 7\}$ and find the second initial conflict-aware schedule $\{\overrightarrow{(6,1)}\}$. Then $H = \{7\}$ and $\overrightarrow{(7,2)}$ is a suitable choice. Thus, the second schedule is $\{\overrightarrow{(6,1)}, \overrightarrow{(7,2)}\}$. Therefore, there are two conflict-aware schedules $\{\overrightarrow{(5,1)}, \overrightarrow{(8,2)}\}$, $\{\overrightarrow{(6,1)}, \overrightarrow{(7,2)}\}$ which also need two time slots. At last, we get finial directional data aggregation tree T_d as shown in Fig.3(b). There are four conflict-aware schedules for the sender set $L(i)$ in level i^{th} and they takes four time slots. However, if we don't adjust the initial directional data aggregation tree G_d, we will construct seven conflict-aware schedules as follows: $\{ \{\overrightarrow{(9,3)}, \overrightarrow{(4,13)}, \overrightarrow{(15,16)}\}, \{\overrightarrow{(10,3)}, \overrightarrow{(17,16)}\}, \{\overrightarrow{(11,3)}, \overrightarrow{(18,16)}\}, \{\overrightarrow{(12,3)}\}, \{\overrightarrow{(5,1)}, \overrightarrow{(8,2)}\}, \{\overrightarrow{(6,1)}\}, \{\overrightarrow{(7,1)}\} \}$, which take seven time slots. We can see that the adjusting procedure could reduce the aggregation schedule latency.

The detail of the conflict-aware scheduling and modulating algorithm is given as follows.

Construct conflict-aware directional data aggregation scheduling. In this subsection, we construct a conflict-free data aggregation scheduling based on above algorithms. After we constructed an initial data aggregation tree G_d, it is obvious that the isolate nodes in NC are leaves in G_d. Therefore, firstly, we schedule all nodes in NC to send data to their parents by algorithm 3. And then, schedule conflict-aware transmissions from the level L^{th}. Obviously, the set of nodes that has not been scheduled after the first step in $L(L)$ is D_L. Suppose $L > 1$. For $\forall 1 \leq i \leq L$, we schedule all nodes in $D_i \subseteq L(i)$ to transmit

Algorithm 2. Sub-DATree-SCH($S, B, G_o, \theta, r, T_d$)

```
Input: A part sender set $S$, a node set $B$, the interference range $r$, the beam-width $\theta$, the connected sensor graph $G_o$ and the adjusting directional data aggregation tree $T_d$.
Output: A conflict-aware sub-schedule $Sch$.
 1: Let $Tag(v)$ denote whether node $v$ has been scheduled as a sender, $E_d$ denote the edge set of the input data aggregation tree.
 2: $A = \emptyset$, $Sch = \{\emptyset\}$
 3: while $S \neq \emptyset$ do
 4:   for each node $v$, $(v \in S)$ do
 5:     if $\forall \overrightarrow{(u, Fa(u))} \in A$, it is not satisfied the following three conditions: (1) If $d(v,u) \leq 1$, then $(\widehat{p_v, \overrightarrow{vu}}) \leq \theta/2$ and $(\widehat{p_u, \overrightarrow{uv}}) \leq \theta/2$; (2) If $d(v, Fa(u)) \leq r$, then $(\widehat{p_v, \overrightarrow{vFa(u)}}) \leq \theta/2$; (3) If $d(u, Fa(v)) \leq r$, then $(\widehat{p_u, \overrightarrow{uFa(v)}}) \leq \theta/2$ then
 6:       $A = A \cup \{(v, Fa(v))\}$, $Tag(v) = 1$, $S = S - \{v\}$
 7:     end if
 8:   end for
 9:   for each node $v$, $(v \in S)$ do
10:     while each node $u$, $u \in (B \cap NG_o(v)) - \{Fa(v)\}$ and $Tag(u) = 0$ do
11:       if $\forall \overrightarrow{(w, Fa(w))} \in A$, it is not satisfied the following three conditions: (1) If $d(v,w) \leq 1$, then $(\widehat{p_v, \overrightarrow{vw}}) \leq \theta/2$ and $(\widehat{p_w, \overrightarrow{wv}}) \leq \theta/2$; (2) If $d(v, Fa(w)) \leq r$, then $(\widehat{p_v, \overrightarrow{vFa(w)}}) \leq \theta/2$; (3) If $d(w,u) \leq r$, then $(\widehat{p_w, \overrightarrow{wu}}) \leq \theta/2$ then
12:         $E_d = \{E_d \cup \{\overrightarrow{(v,u)}\}\} - \{\overrightarrow{(v, Fa(v))}\}$, $Fa(v) = u$,$A = A \cup \{(v, Fa(v))\}$, $Tag(v) = 1$, $S = S - \{v\}$
13:       end if
14:     end while
15:   end for
16:   $Sch = Sch \cup \{A\}$, $A = \emptyset$
17: end while
18: return $Sch$
```

data to their parents in $L(i-1)$. Nextly, we schedule nodes in $C_i \subseteq L(i-1)$ to transmit data to their parents in $L(i-1) \cup L(i-2)$. Repeat these steps till $i = 1$. Here, $L(0) = \{s\}$, it is just need to schedule the nodes in D_1 to transmit to sink s. For each level $i(1 < i < L)$, we first schedule the nodes in C_{i+1} to their parents which are in $L(i)$ and $L(i-1)$ and $Tag()$ is 0. So after the scheduling and adjusting procedure, all nodes in C_{i+1} transmit data to other nodes in the level i^{th} and $(i-1)^{th}$, which are not still been scheduled. After NC and C_{i+1} are scheduled, we schedule the nodes in D_i to their patents in $L(i-1)$. Therefore, all nodes in $L(i)$ has been scheduled. After running these procedures, the sink could receive the data from all nodes in the network since $\bigcup_{i=1}^{L}(D_i \cup C_i) \cup NC = V$. Consequently, the number of the elements in A_{sch} is the latency l of this algorithm.

The scheduling algorithm for the LMASD problem is presented as follows:

Algorithm 3. Final-DATree-SCH(G_o, s, θ, r)

```
Input: A connected sensor network $G_o = (V, E)$, a sink node $s (s \in V)$, the beam-
    weigh $\theta$, and interference range $r$.
Output: A conflict-aware directional data aggregation scheduling $A_{sch}$.
 1: Let $T_d = (V, E_d)$ denote the finial directional data aggregation tree, and $Tag(v)$
    denote whether node $v$ is scheduled as a sender in $T_d$.
 2: for each node $v \in S$ do
 3:     $Tag(v) = 0$
 4: end for
 5: $A_{sch} = \{\emptyset\}$
 6: $[T_{BFS}, \{L(0), L(1), \cdots, L(L)\}, D, \{D_0, D_1, \cdots, D_L\}] \leftarrow$ BFS-Tree($G_o, s$)
 7: $[G_d, \{C_1, C_2, \cdots, C_L\}, NC] \leftarrow$ Initial-DA-Tree($G_o, T_{BFS}, D, \{D_0, D_1, \cdots, D_L\}$)
 8: $T_d \leftarrow G_d$
 9: $A_{sch} \leftarrow A_{sch} \cup$ Sub-DATree-SCH($NC, V, G_o, \theta, r, T_d$)
10: for $i = L$ to 1 do
11:     $A_{sch} \leftarrow A_{sch} \cup$ Sub-DATree-SCH($D_i, L(i-1), G_o, \theta, r, T_d$)
12:     if $i = 1$ then
13:         Break;
14:     end if
15:     $A_{sch} \leftarrow A_{sch} \cup$ Sub-DATree-SCH($C_i, L(i-1) \cup L(i-2), G_o, \theta, r, T_d$)
16: end for
17: return $A_{sch}$
```

5 Performance Analysis

In this section, we study the time complexity and the approximation ration of Algorithm 4. We will compare the performance of our algorithm against the trivial lower bound of L, where L is the depth of the BFS tree T_{BFS}.

5.1 Time Complexity

Theorem 2. *The time complexity of algorithm* 3 *is* $O(|V|^3)$.

Proof: Suppose that $|V|$ is the number of nodes of the omni-directional network topology $G_o = (V, E)$, and Δ is the maximum degree of the nodes in $G_o(v) (v \in V)$.

(1) It is easy to know that the constructions of BFS and MIS both have time complexity $O(|V|^2)$. Therefore, Step 6 takes time $O(|V|^2)$.

(2) Step 7 in Algorithm 3 has time complexity $O(|V|^2)$. To construct an initial directional data aggregation tree, each node needs to check at most Δ nodes to find a connector or dominator. Therefore, Step 7 takes time $O(|V| \times \Delta) \leq O(|V|^2)$.

(3) Step 9 to 16 in Algorithm 3 has time complexity $O(|V|^3)$. Suppose that there are $|W|$ directed edges from senders in sender set $S(|S| = W)$ to their parents in the initial directed data aggregation tree G_d. Firstly, we schedule these directed edges to transmit simultaneously. Each directed edge needs to

check at most $|W|-1$ directed edges to confirm whether they can be scheduled at the same time slot. This procedure takes time $O(|W||W-1|) = O(|W|^2)$. Secondly, we adjust the initial directed data aggregation tree. Each node which is not be scheduled in S needs to change its parent to one of its neighbors and check whether this adjusting directed edge conflict with the current schedule. This procedure takes time $O(|W| \times (\Delta - 1) \times |W-1|) = O(|W|^2 \times \Delta)$. To sum up the two procedures,there are time complexity $O(|W|^2 \times \Delta)$. Therefore, steps 9 to 16 in algorithm 4 take time $O(|W_1|^2 \times \Delta + O(|W_2|^2 \times \Delta) + O(|W_3|^2 \times \Delta) + \cdots) = O(|E_d|^2 \times \Delta) \leq O(|V|^3)$, since $|W_1| + |W_2| + \cdots = |E_d| = |V| - 1$, where $|E_d|$ is the number of directed edges in finial directional data aggregation tree T_d.

To sum up these analysis above, Algorithm 3 has time complexity $O(|V|^3)$.

5.2 Algorithm Approximation Ratio

In this subsection, we study the approximation ratio of Algorithm 3. Before analyzing the performance, we need to introduce a lemma which will be used to analyze upper bound of our scheduling latency. In this Lemma, $\theta = 2\pi$, and the transmission range, the interference range and the carrier sensing range, respectively is r, αr and βr.

Lemma 2. *[15] In order for two parallel transmissions "$t_1 \rightarrow r_1$" and "$t_2 \rightarrow r_2$" to be non-conflicting according to the network model, it is sufficient to have: $d(t_1, t_2) > max(\alpha + 1, \beta)r \vee d(r_1, r_2) > (max(\alpha, \beta) + 2)r$.*

Our interference model is same as [15] when $r = 1$, $\alpha = r_d$ and $\beta = 1$. If we set the beam-width of directional antenna $\theta = 2\pi$, we could get our sufficient conditions.

Lemma 3. *In order for two parallel transmissions "$a \rightarrow b$" and "$c \rightarrow d$" to be non-conflicting according to the network model, it is sufficient to have: $d(a, c) > (r_d + 1) \vee (b, d) > (r_d + 2)$.*

We construct two conflict graphs GC_t and GC_r for the omni-directional connected graph $G_o = (V, E)$ according to Lemma 3. The graphs GC_t and GC_r both have the same vertices as the graph G_o and graph GC_t has an edge between two nodes v and u if and only if $d(v, u) \leq r_d + 1$, graph GC_t has an edge between two nodes v and u if and only if $d(v, u) \leq r_d + 2$. For example, two parallel transmissions "$a \rightarrow b$" and "$c \rightarrow d$" are non-conflicting if they are sufficient to have $(a, c) \notin E(GC_t)$ or $(b, d) \notin E(GC_r)$. If there is no edge between node v and u $(v, u \in V)$ in the two graph, then the two nodes can not conflict with each other in the initial directional data aggregation tree T_d.

Next, we introduce the following lemma and Definition 1.

Lemma 4. *[17]The area of the convex hull of any $n \geq 2$ non-overlapping unit-radius circular disks is at least $2\sqrt{3}(n-1) + (2-\sqrt{3})\lceil\sqrt{12n-3} - 3\rceil + \pi$.*

Definition 2. *[13] $f(x)$ is the maximum integer n to make $2\sqrt{3}(n-1) + (2-\sqrt{3})\lceil\sqrt{12n-3} - 3\rceil + \pi \leq x$.*

Here, the value of $f(x)$ is a constant integer obtained easily for a given x.
Using the similar method as [13] , we can get the following lemma.

Lemma 5. *The inductivities of $GC_t[S]$ and $GC_r[S]$ have an upper-bound of $\tilde{A} = f(2\pi h^2 + 2\pi h - 4h)$ and $\tilde{B} = f(2\pi k^2 + 2\pi k - 4k)$ respectively, where $h = r_d + 1$ and $k = r_d + 2$.*

Theorem 3. *The latency upper bound of algorithm 3 is at most $(\tilde{a} + 19\tilde{b})L + \Delta\tilde{b} - \tilde{a} - 19\tilde{b} + 5$, where $\tilde{a} = 1 + f(2\pi h^2 + 2\pi h - 4h)$, $\tilde{b} = 1 + f(2\pi k^2 + 2\pi k - 4k)$, Δ and L are respective the maximum degrees of omni-directional connected graph G_o and the depth of the BFS tree T_{BFS}, $h = r_d + 1$ and $k = r_d + 2$. Here Δ contributes to an additive factor instead of a multiplicative one, thus Algorithm 3 has a nearly approximation ratio $(\tilde{a} + 19\tilde{b})$.*

Proof: In order to prove this theorem, we need to calculate an upper bound on the number of time slots consumed by each subschedule. We consider the following 3 parts:

Part 1: Sub-DATree-SCH $(NC, V, G_o, \theta, r, T_d)$. This part only is done once for the whole schedule. In the first procedure of the schedule, we find all the conflict-aware transmissions in initial directional data aggregation tree G_d. Suppose H to represent the parent nodes set of NC. Here, we first consider the beam-width of directional antenna $\theta = 2\pi$ and the conflicting graph $GC_r[H]$ constructed by the undirected connect graph G_o. Note that, for each node in NC, its parent in H is the dominator node, which is the independent node included in the independent set D constructed from $BFS(G_o, s)$. For each node in H, it has at most Δ neighbors, and at most $\Delta - 1$ neighbors need to transmit data to it. Since the nodes in H are independent and scheduled to receive data, we can acquire a proper node coloring of nodes in H which can be accomplished with $1 + \tilde{B}$ colors according to Lemma [13], with the upper bound $\tilde{B}$ of the inductivity of the graph $GC_r[H]$. This means that the nodes with the same color can be scheduled to receive data in the same time slot. Therefore, the subschedule can be finished within $(\Delta - 1)(1 + \tilde{B})$ time slots. However, there are more conflict-free transmissions for each schedule using directional antenna $\theta < 2\pi$. Moreover, in the second procedure of the schedule in Algorithm 3, we can find more sender nodes to transmit data for each schedule as the example shown in Fig.3 and also gain more conflict-free transmissions for each schedule by the adjusting the parents of the senders which are not scheduled. Consequently, the Sub-DATree-SCH $(NC, V, G_o, \theta, r_d, T_d)$ is guaranteed to finished at most $(\Delta - 1)(1 + \tilde{B})$ time slots.

Part 2: Sub-DATree-SCH $(D_i, L(i-1), G_o, \theta, r, T_d)$, $\forall 1 \le i \le L$. For D_i, $\forall 2 \le i \le L$, all nodes in D_i are independent and need to transmit data to their parents. With the upper bound $\tilde{A}$ of inductivity of $GC_t[D_i]$, nodes in D_i has a proper node coloring using at most $1 + \tilde{A}$ colors according to Lemma [13]. Here, since the nodes with same colors can be scheduled to transmit simultaneously, the subschedule is guaranteed to finish within $1 + \tilde{A}$ time slots. For D_1, note that the parent set of all nodes in D_1 is $H = C_1 = \{s\}$, where s is the sink node.

There are at most 5 independent nodes in unit disk graph, that is $|D_1| \leq 5$, so this subschedule can be finished in 5 time slots. However, with the directional antenna $\theta < 2\pi$ and the adjusting process in the second procedure previously analyzed, the time slots of schedule maybe less than $1 + \tilde{A}$, for $2 \leq i \leq L$. Consequently, the Sub-DATree-SCH $(D_i, L(i-1), G_o, \theta, r_d, T_d)$ is guaranteed to finish in $1 + \tilde{A}$ time slots for $2 \leq i \leq L$ and 5 time slots when $i = 1$.

Part 3: Sub-DATree-SCH $(C_i, L(i-1) \cup L(i-2), G_o, \theta, r_d, T_d)$, $\forall 2 \leq i \leq L$. For the conflict graph $GC_r[H]$ where H is the parent set of C_i, each node in C_i has to transmit data to its parent in H, where H is a subset of independent set D. By [3], each node in H has 19 neighbors in $C_i (3 \leq i \leq L)$ and 20 neighbors in C_2. Since they are mutually independent and are scheduled to receive data, nodes in H has a proper node coloring using at most $1 + \tilde{B}$ colors according to Lemma [13], with the upper bound $\tilde{B}$ of inductivity of $GC_r[H]$. This means that the nodes with the same color can be scheduled to receive data in the same time slot. Therefore, the subschedule can be finished within $19(1 + \tilde{B})$ time slots for $3 \leq i \leq L$ and $20(1 + \tilde{B})$ time slots when $i = 2$. However, with the directional antenna $\theta < 2\pi$ and the adjusting process in the second procedure, the time slots of schedule must be less than the time slots of their respective situations above. Therefore, the Sub-DATree-SCH $(C_i, L(i-1) \cup L(i-2), G_o, \theta, r, T_d)$ is guaranteed to finish in $19(1 + \tilde{B})$ time slots for $3 \leq i \leq L$ and $20(1 + \tilde{B})$ time slots when $i = 2$.

To sum up above, Algorithm 3 takes at most $(\Delta - 1)(1 + \tilde{B}) + (L-1)(1 + \tilde{A}) + 5 + (L-2) \times 19(1 + \tilde{B}) + 20(1 + \tilde{B})$ time slots. To make it clear, let $\tilde{a} = 1 + \tilde{A}$ and $\tilde{b} = 1 + \tilde{B}$. Algorithm 3 has a nearly constant $(\tilde{a} + 19\tilde{b})$ ratio.

Acknowledgment. This research was jointly supported in part by the National Natural Science Foundation of China under grant 61070191, the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China under Grant 10XNJ032.

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A Survey on Sensor Networks. IEEE Communications Magazine 40(8), 102–114 (2002)
2. Chen, X., Hu, X., Zhu, J.: Minimum Data Aggregation Time Problem in Wireless Sensor Networks. In: Jia, X., Wu, J., He, Y. (eds.) MSN 2005. LNCS, vol. 3794, pp. 133–142. Springer, Heidelberg (2005)
3. Huang, S.C.-H., Wan, P.-J., Vu, C.T., Li, Y.-S., Yao, F.: Nearly Constant Approximation for Data Aggregation Scheduling in Wireless Sensor Networks. In: IEEE INFOCOM 2007, pp. 366–372 (2007)
4. Shang, W., Wan, P.-J., Hu, X.: Approximation Algorithm for Minimal Convergecast Time Problem in Wireless Sensor Networks. Wireless Networks 16(5), 1345–1353 (2010)
5. Wan, P.-J., Huang, S.C.-H., Wang, L.X., and Jia, X. H.: Minimum-latency Aggregation Scheduling in Multihop Wireless Networks. In: ACM MOBIHOC 2009, pp. 185-194 (2009)

6. Wan, P.-J., Wang, Z., Wan, Z.Y., Huang, S.C.-H., Liu, H.: Minimum-latency schedulings for group communications in multi-channel multihop wireless networks. In: Liu, B., Bestavros, A., Du, D.-Z., Wang, J. (eds.) WASA 2009. LNCS, vol. 5682, pp. 469–478. Springer, Heidelberg (2009)
7. Guo, S., Yang, O., Leung, V.: Approximation algorithms for Longest-lived Directional Multicast Communications in WANETs. In: MobiHOC 2007, pp. 190–198 (2007)
8. Kathiravan, K., Selvi, S.T., Reshmi, R.: Efficient Broadcast in MANETS using Directional Antennas. Ubiquitous Computing and Communication Journal 2(2), 1–7 (2007)
9. Li, D., Li, Z., Liu, L.: Energy efficient broadcast routing in ad hoc sensor networks with directional antennas. In: Li, Y., Huynh, D.T., Das, S.K., Du, D.-Z. (eds.) WASA 2008. LNCS, vol. 5258, pp. 29–39. Springer, Heidelberg (2008)
10. Yang, S., Wu, J.: Efficient Broadcasting using Network Coding and Directional Nntennas in MANETs. IEEE Transaction on Papallel and Distrbuted Systems 21(2), 148–161 (2010)
11. Wan, P.-J., Xu, X., Wang, L., Jia, X., Park, E.K.: Minimum-lantency Beaconing Schedule in Multihop Wireless Networks. In: IEEE INFOCOM 2009, pp. 2340–2346 (2009)
12. Yu, B., Li, J., Li, Y.: Distributed Data Aggregation Scheduling in Wireless Sensor Networks. In: IEEE INFOCOM 2009, pp. 2159–2167 (2009)
13. Zhu, Q., Li, D.: Approximation for a Scheduling Problem with Application in Wireless Networks. SCIENCE CHINA Mathematics 53(6), 1643–1655 (2010)
14. Lee, S.-H., Lee, K.-W., Cho, Y.-Z.: Directional Flooding Scheme with Data Aggregation for Energy-efficient Wireless Sensor Networks. In: WASA 2008, pp. 821–830 (2009)
15. Mahjourian, R., Chen, F., Taiwari, R.: An Approximation Algorithm for Comflict-aware Broadcast Scheduling in Wireless Ad Hoc Networks. In: MOBIHOC 2008, pp. 331–340 (2008)
16. Wegner, G. : ÜberEndliche Kreispackungen in Der Ebebe. Studia Sci. Math. Hungar, pp. 1-28 (1986)
17. Matula, D.W., Beck, L.L.: Smallest-last Ordering and Clustering and Graph Coloring Algorithms. In: ACM 1989, pp. 417–427 (1989)

A Near-Optimal Memoryless Online Algorithm for FIFO Buffering Two Packet Classes

Fei Li[⋆]

Department of Computer Science, George Mason University
Fairfax, VA 22030, USA
http://www.cs.gmu.edu/~lifei

Abstract. We consider scheduling packets with values in a capacity-bounded buffer in an online setting. In this model, there is a buffer with limited capacity B. At any time, the buffer cannot accommodate more than B packets. Packets arrive over time. Each packet is associated with a non-negative value. Packets leave the buffer only because they are either sent or dropped. Those packets that have left the buffer will not be reconsidered for delivery any more. In each time step, at most one packet in the buffer can be sent. The order in which the packets are sent should comply with the order of their arrival time. The objective is to maximize the total value of the packets sent in an online manner. In this paper, we study a variant of this FIFO buffering model in which a packet's value is either 1 or $\alpha > 1$. We present a deterministic memoryless 1.304-competitive algorithm. This algorithm has the same competitive ratio as the one presented in (Lotker and Patt-Shamir. PODC 2002, Computer Networks 2003). However, our algorithm is simpler and does not employ any marking bits. The idea used in our algorithm is novel and different from all previous approaches applied for the general model and its variants. We do not proactively preempt one packet when a new packet arrives. Instead, we may preempt more than one 1-value packet when the buffer contains sufficiently many α-value packets.

1 Introduction

We consider online algorithms to schedule packets with values in a capacity-bounded buffer. There is a buffer with a limited size $B \in \mathbb{Z}^+$. At any time, the buffer can accommodate at most B packets. Packets arrive over time. The buffer is preemptive: Packets already in the buffer are allowed to be dropped at any time before they are delivered. We use $r_p \in \mathbb{R}^+$ and $v_p \in \mathbb{R}^+$ to denote the *release time* (*arriving time*) and *value* of a packet p respectively. Packets leave the buffer only because they are either sent or dropped. Those sent and dropped packets will not be reconsidered for delivery any more. Time is discrete. In each time step, at most one packet in the buffer can be sent. The order of the packets being sent should comply with the order in which they are released. The objective is to maximize the total value of the packets sent in an online manner.

[⋆] Research partially supported by NSF Grant CCF-0915681.

We call this model a *FIFO buffering model*; this model has attracted a lot of attention in the past ten years and has been studied extensively [4][8][6][3]. In this paper, we study a variant of this model in which packets have value either 1 or $\alpha > 1$. This variant is called a *two-valued model* and has been investigated in [5][8][3].

Without knowing the future input, an online algorithm has to make decision over time based on the input information that it has seen so far. If an online algorithm decides which packet to send only based on the contents of its current buffer, and independent of the packets that have already been released and processed, we call it *memoryless*. Consider a maximization problem as an example. A deterministic online algorithm is called *k-competitive* if its objective value on *any* instance is at least $1/k$ times of the objective of an optimal offline algorithm applied on the same instance [1]. The *upper bounds* of competitive ratio are achieved by some known online algorithms. A competitive ratio less than the *lower bound* cannot be reached by any online algorithm [1]. For the two-valued model, the previously best known result is a 1.544-competitive memoryless algorithm [5], a 1.304-competitive algorithm [8] using marking bits to associate with all pending packets in the buffer [8], and a non-memoryless optimal 1.282-competitive algorithm [3]. In this paper, we present a 1.304-competitive memoryless algorithm. Our algorithm is simpler than the one in [8] and it does not use marking bits.

It is instructive to compare and contrast the algorithm in [8] with ours since both have the same competitive ratio 1.304. Based on the definition of *memoryless algorithms* [2][7] (the algorithm should make the decisions independent of any packets that it has processed), we know that the marking bits used by [8] reflect the packets that the algorithm has processed and affect the marking and flush procedure for later arriving α-value packets. Hence the algorithm in [8] is not memoryless.

In Section 2, we describe a deterministic memoryless online algorithm called ON. In Section 3, we give the algorithm's analysis, showing that it is 1.304-competitive. Related work and conclusion remarks are presented in Section 4.

2 Algorithm

Without loss of generality, we assume all packets have distinct release time. Consider m packets released in the same time step t. We let these m packets have distinct release time of t, $t+\delta$, $t+2\delta$, …, $t+(m-1)\delta$ respectively, where $\delta > 0$ and $m \cdot \delta \leq 1$, in the order of being released.

2.1 The Idea

The greedy approach might be the first intuitive method to design competitive online algorithms for the FIFO buffering model. It works as follows. If packets overflow, the minimum-value packet is dropped (with ties broken arbitrarily). In each time step, the earliest released packet in the buffer is sent. The greedy algorithm is asymptotically no better than 2-competitive for the FIFO buffering model, even for the two-valued variant. Based on the observation from the tight instance for the greedy approach, Kesselman et al. in [6] came up with another idea, which is to proactively preempt the 1-value packets in the buffer released before the α-value packets. Consider a 1-value packet p and an α-value packet q with $r_p < r_q$. On one hand, if q but not p is the packet sent by the optimal offline algorithm, we would like to preempt p proactively at q's arrival and expect that q can be sent before packet overflow happens. On the other hand, if both p and q are sent by the optimal offline algorithm, we would like to preempt p only if p's value is bounded by a fraction of q's value. This idea leads to the memoryless algorithm PG, which is 1.732-competitive for the general case [3] and 1.544-competitive for the two-valued setting [5].

In algorithm PG, a packet p's preemption is due to buffering another packet q. A new arrival preempts at most one packet that is released earlier. Motivated by the greedy algorithm and PG, we propose the following strategy:

Solution 1. We preempt a set of 1-value packets due to the existence of a set of α-value packets in the buffer to make room for the potential α-value packets that are released later.

Different from PG, we take into account the values of multiple packets to preempt a packet. Based on this idea, a 1-value packet is preempted only when the buffer has buffered sufficiently many of later-released α-value packets. In addition, multiple 1-value packets may be preempted at the arrival of one α-value packet.

2.2 A Memoryless Online Algorithm for the Two-Valued Model

We name our algorithm ON. ON represents a family of deterministic memoryless online algorithms parameterized by a real number $\beta > 0$. Denote Q_t^{ALG} as the buffer of an algorithm ALG at time t. Without confusion, we may omit the subscript t in our notation.

Definition 1 (Ejectable Packet). *Consider two packets p and q in the buffer with $r_p < r_q$, $v_p = 1$ and $v_q = \alpha$. Such a packet p may prevent us from sending q before a future possible packet overflow, and we call p an* ejectable packet.

Algorithm ON is outlined as follows. New packets are admitted in a greedy manner. If the earliest-released packet in the buffer is an α-value packet, we simply send this packet, same as the greedy policy. Otherwise, if the earliest-released packet is a 1-value packet, we preempt all the ejectable packets, if the total value of all the ejectable packets is bounded by $1/\beta$ times of the total value of all the α-value packets in ON's buffer. Then the earliest-released packet, which is a 1-value packet if no preemption happens or an α-packet if preemption occurs, is sent. In each time step t, ON is described in two stages: *admitting packets* (see Algorithm 1) and *(possibly) preempting* 1*-value packets and delivering a packet* (see Algorithm 2).

Algorithm 1. Admitting Packets

```
1: for each new arriving packet do
2:     if there is a buffer slot available then
3:         append this packet at the end of the packet queue;
4:     else
5:         evict the minimum-value packet, with ties broken in favor of the earliest-
           released packet.
6:     end if
7: end for
```

Algorithm 2. Preempting Packets and Delivering a Packet

```
1: Let the earliest-released packet in the buffer be $e$.
2: if $v_e = \alpha$ then
3:     send $e$;
4: else
5:     define $D := \{p \mid p \in Q^{\mathrm{ON}},\ v_p = 1,\ \exists q$ with $v_q = \alpha$ and $r_p < r_q\}$;
6:     if $\sum_{q \in Q^{\mathrm{ON}}; v_q = \alpha} v_q \geq \beta \sum_{p \in D} v_p$ then
7:         preempt all the packets in $D$;
8:     end if
9:     send the earliest-released packet in the buffer.
10: end if
```

Example 1. Let $B = 3$ and $\beta = \alpha$. We use $(r_p,\ v_p)$ to denote a packet p with release time r_p and value v_p. Remember that we use fractional release time to differentiate those packets released at the same time step. An input instance is given as follows.

$$\begin{aligned}
&\text{step 1}: \ (1,\ 1),\ (1.1,\ 1),\ (1.2,\ \alpha)\\
&\text{step 2}: \ (2,\ \alpha),\ (2.1,\ \alpha),\ (2.2,\ \alpha),\ (2.3,\ 1)\\
&\text{step 3}: \ \text{no released packets}\\
&\text{step 4}: \ \text{no released packets}\\
&\text{step 5}: \ (5,\ 1),\ (5.1,\ \alpha),\ (5.2,\ \alpha)
\end{aligned}$$

The optimal offline policy OPT sends the following packets in order:

$$(1.2,\ \alpha),\ (2,\ \alpha),\ (2.1,\ \alpha),\ (2.2,\ \alpha),\ (5,\ 1),\ (5.1,\ \alpha),\ (5.2,\ \alpha).$$

ON sends packet $(1,\ 1)$ in the first time step without preempting the ejectable packets. ON admits all the α-value packets released in step 2 and sends them in steps 2, 3, and 4. In step 5, ejectable packet $(5,\ 1)$ is preempted. ON sends packets $(5.1,\ \alpha)$ and $(5.2,\ \alpha)$ consecutively in steps 5 and 6. Finally, ON has the following sequence of packets being sent.

$$(1,\ 1),\ (2,\ \alpha),\ (2.1,\ \alpha),\ (2.2,\ \alpha),\ (5.1,\ \alpha),\ (5.2,\ \alpha).$$

For the above instance, OPT and ON gain the total values of $6\alpha + 1$ and $5\alpha + 1$, respectively.

3 Analysis

Theorem 1. *ON is* $\max\{\frac{1+\beta}{\beta},\ \frac{\alpha^2+2\alpha\cdot\beta}{\alpha^2+\alpha\cdot\beta+\beta}\}$*-competitive, where* $\beta > 0$.

We will employ a charging scheme to prove Theorem 1. Let OPT denote an optimal offline algorithm and $\mathcal{O}$ denote the set of packets sent by OPT.

Lemma 1. *Any* α*-value packet that ON sends is an* $\mathcal{O}$*-packet.*

Proof. Assume there exists an α-value packet $p \notin \mathcal{O}$ that is sent by ON. Using an exchange argument, we will show that there must exist another optimal offline algorithm that sends p.

Consider an algorithm MOPT (Modified OPT) which admits packets $\mathcal{O} \cup \{p\}$ and sends the earliest-released packet in the buffer in each time step. From step 1 to step r_p, MOPT's buffer content and the packet it sends in each time step are the same as those of OPT. We claim that packet overflow must happen in MOPT's buffer at some time step at/after time r_p. Otherwise, MOPT can send all the packets in $\mathcal{O} \cup \{p\}$ successfully and gains more than OPT, which contradicts the fact that OPT is optimal. Assume $t \geq r_p$ is the first time at which packet overflow occurs in MOPT's buffer. Since MOPT only accepts one more packet p in addition to the packets $\mathcal{O}$, there are $(B+1)$ packets for MOPT to buffer at time t. We simply drop one packet $q \neq p$ out of MOPT's buffer at time t. Because there is no packet overflow in the time interval $[1,\ t]$ and the number of packets buffered by MOPT (after we drop q) is the same as that of OPT's at any time after time t, MOPT is capable of sending all the packets $\mathcal{O} \cup \{p\} \setminus \{q\}$ in a FIFO order and gains a total value $\geq \sum_{j\in\mathcal{O}} v_j$. □

The contrapositive of Lemma 1 leads to the following corollary.

Corollary 1. *Any non-*$\mathcal{O}$*-packet that ON sends is a* 1*-value packet.*

Remark 1. From Algorithm 2, no α-value packets can be preempted. That is, any unsent α-value packet must have been only evicted by ON.

Remark 2. Consider a time t in which an α-value packet is evicted by ON. This must indicate that the current ON's buffer is full of B packets with value α. From Algorithm 2, these B packets with value α will be sent by ON in steps $t,\ t+1,\ \ldots,\ t+B-1$.

Remark 3. From Algorithm 2, if ON preempts some 1-value packets in a step t, ON will send all the preempting α-packets in the following time steps. These preempting packets have a total value of at least β times of those preempted 1-value packets.

In the following, we introduce our charging scheme. Because it is difficult to compare ON with OPT directly, we compare ON with a relaxed algorithm called ROPT. We will show that ROPT gains the same total value as OPT does. In our charging scheme, we will charge values to ROPT and ON. Algorithm ROPT's operations at a time step t is outlined in Algorithm 3.

Algorithm 3. Relaxed OPT (p, t)

```
1: Accept each $\mathcal{O}$-packet arriving at step $t$.
   {In Lemma 2, we prove that all the $\mathcal{O}$-packets can be admitted by ROPT without
   encountering overflow.}
   {Let $p$ be the packet that ON sends in $t$. If ON sends nothing in $t$, $p$ is defined as
   a non-$\mathcal{O}$ null packet with value 0.}
2: if $p \in \mathcal{O}$ and $p$ is in ROPT's buffer then
3:    send $p$;
4: else
5:    send the earliest-released packet in the buffer, if any.
6: end if
```

Lemma 2. *All $\mathcal{O}$-packets are accepted by ROPT.*

Proof. To prove Lemma 2, we only need to show that at any time ROPT's buffer contains no more pending packets than OPT's buffer, and thus packet overflow does not happen to ROPT when admitting $\mathcal{O}$-packets. We apply the induction method. Initially, ROPT's and OPT's buffers are empty. Assume at time t, the number of packets in ROPT's buffer is no more than the number of packets in OPT's buffer. In step t, ROPT sends one packet, if any, and OPT sends one packet, if any. Then after packet delivery, ROPT's buffer still contains no more packets than OPT's buffer. □

Corollary 2. *ROPT sends all the O-packets.*

Given Lemma 2 and the fact that OPT successfully sends all the $\mathcal{O}$-packets, Corollary 2 easily holds. Lemma 2 and Corollary 2 guarantee that

Remark 4. In our charging scheme design, we only need to compare ON to ROPT instead of to OPT.

We describe an important observation of ROPT in Remark 5.

Remark 5. For any $\mathcal{O}$-packet p that is sent by ON in step t, ROPT either has sent p before t or sends p in the same step t.

Definition 2 (Chain of Steps). *For a chain consisting of k time steps*

$$c_1 \rightarrow c_2 \rightarrow \cdots \rightarrow c_k,$$

where $c_1 < c_2 < \cdots < c_k$, ON sends a non-$\mathcal{O}$-packet in step c_1 and for all $i = 1, \ldots, k-1$, the packet that ROPT sends in step c_i is the packet that ON sends in step c_{i+1}. Note that these time steps do not need to be successive. Chains do not share time steps.

Lemma 3. *At any time, for any $\mathcal{O}$-packet in ON's buffer but not in ROPT's buffer, there is a unique corresponding chain of steps.*

Algorithm 4. Construction of a Chain of Steps (t)

```
1: From Remark 5, there exists a unique previous time step $time(p) < t$ in which
   ROPT sends $p$ and ON sends another packet $q \neq p$.
2: if $q \notin \mathcal{O}$ then
3:    create a chain of steps consisting of only one time step $time(p)$.
4: else {that is, $q \in \mathcal{O}$}
5:    construct a chain of steps $time(q) \rightarrow time(p)$;
      {From Remark 5, ROPT must send $q$ in a unique time step $time(q) < time(p)$.}
6:    while the packet $q'$ that is sent by ON in $time(q)$ is an $\mathcal{O}$-packet do
7:       expand the chain by inserting $time(q')$ to the front of the current chain;
8:       $q$ is replace by $q'$ (for ease of notation of while loop);
9:    end while
10:   expand the chain by inserting $time(q')$ to the front of the current chain and this
      chain is completed.
      {We have found a non-$\mathcal{O}$-packet sent by ON and thus the chain is completed, as
      the head of the chain has to be a non-$\mathcal{O}$-packet.}
11: end if
```

Proof. In Algorithm 4, we introduce how to build up a chain of steps for each $\mathcal{O}$-packet p in ON's buffer but not in ROPT's buffer at time t. This construction directly proves Lemma 3. We use $time(p)$ to denote the time step in which ROPT sends a packet p. □

The basic idea is to start building the chain from the end to the head, in the reverse order of time. The end step is a step that an $\mathcal{O}$-packet is in ON's buffer but not in ROPT's buffer. Starting from this step, we search backwards in time to look for the step in which ROPT sends this $\mathcal{O}$-packet. From Remark 5, we know that such a step must proceed the end step. Then we look at the packet sent by ON in this step, if it is a non-$\mathcal{O}$-packet, we stop constructing the chain because we have found the head step of the chain. If it is an $\mathcal{O}$-packet, we expand the chain and continue to search backwards until we find a non-$\mathcal{O}$-packet sent by ON.

A characteristics of a chain is that in each time step in the chain except for the first step, ON sends an $\mathcal{O}$-packet.

Figure 1 demonstrates the construction of the chain of steps. Three cases are given. The packets selected by ON and ROPT to send in each step are shown. Capital letter packet is a non-$\mathcal{O}$-packet, and small letter packet is an $\mathcal{O}$-packet. The corresponding chains are plotted on the right.

In the following, we introduce our charging scheme for both ON and ROPT. The charging scheme will use the procedure of constructing chains. For those $\mathcal{O}$-packets that are preempted or evicted by ON, we construct the chains at the time steps when they are preempted or evicted. Then, we charge these $\mathcal{O}$-packets' values to ROPT either at the first time steps of the chains or in the time steps after they are preempted or evicted. Details of the charging scheme are described as follows.

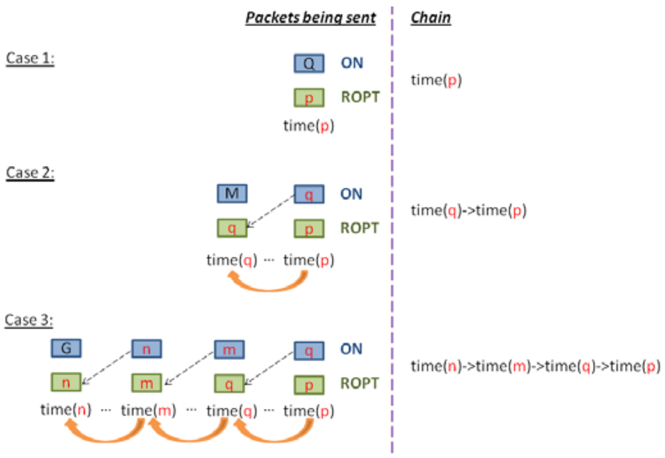


Fig. 1. The construction of chains of steps

Definition 3. Open/closed chain. *Given a time $t \geq c_k$, consider a chain of k time steps $c_1 \rightarrow c_2 \rightarrow \cdots \rightarrow c_k$. We call this chain* closed *if we have charged the value of a packet in ROPT's current buffer to ROPT in step c_1. (Note that in step c_1, ON sends a non-$\mathcal{O}$-packet; from Corollary [1], this packet is a* 1*-value packet.) Otherwise, we say that this chain is* open*.*

Case 1. For each packet sent by ON, we charge ON the value of this packet in the time step that it is being sent.

Case 2. For those $\mathcal{O}$-packets that are sent by both ROPT and ON, we charge their values to ROPT in the time steps that ON sends them.

Case 3. For the $\mathcal{O}$-packets that are not sent by ON, they are either evicted or preempted.

Assume the unsent packet is an α-value packet. Those α-value $\mathcal{O}$-packets that are not delivered by ON can only be evicted (from Remark [1]).

Let p denote an evicted α-value $\mathcal{O}$-packet and d_p denote the time in which ON evicts p. From Remark [2], ON sends at least B packets with value α after time d_p. We define l_p as the first time step after d_p such that in step l_p ON sends either a 1-value packet or nothing. We charge the value $v_p = \alpha$ to ROPT in the time interval $[d_p,\ l_p - 1]$.

Assume the unsent packet is a 1*-value packet.* Those 1-value $\mathcal{O}$-packets that are not delivered by ON can be evicted or preempted by ON.

1. Assume p is a preempted 1-value $\mathcal{O}$-packet and d_p is the time in which ON preempts p.

   (a) Assume there is an open chain associated with a preempting packet at time d_p.
   Let q be the earliest-released α-value preempting packet in ON's buffer whose corresponding chain is open. From Algorithm 4, q is an $\mathcal{O}$-packet that has been sent by ROPT in a previous time step $< d_p$. In the first step of this chain, say t', ON sends a 1-value non-$\mathcal{O}$-packet (see Corollary 1).
   We charge the value $v_p = 1$ to ROPT in step t' and we close this chain.
   (b) Assume there is no open chain associated with any preempting packet at time d_p.
   Let $h \geq 1$ be the number of preempting packets in ON's buffer at time d_p. From Remark 3, ON sends these preempting α-packets consecutively from d_p to $d_p + h - 1$.
   We charge the value $v_p = 1$ to ROPT in each of the time step in the interval $[d_p,\ d_p + h - 1]$.
2. Assume p is an evicted 1-value $\mathcal{O}$-packet and d_p is the time in which ON evicts p.
   (a) Assume p is an $\mathcal{O}$-packet that has been sent by ROPT by time step d_p. From Lemma 3, p corresponds to a chain of steps such that p is the packet sent by ROPT in the last time step of the chain and in the first time step of the chain, say, t', ON sends a 1-value non-$\mathcal{O}$-packet (see Corollary 1). Because $v_p = 1$, p is not a preempting packet and no preempted 1-value packet has been charged in the first step t' of the chain for ROPT.
   We charge the value $v_p = 1$ to ROPT in step t' and we close this chain.
   (b) Assume p is rejected by ON at its arrival. ROPT accepts p and will send p in a later time step $\geq d_p$.
   From Algorithm 1, if this case happens, it must be true that ON's buffer is full of α-value packets at p's arrival.
   We first claim that all the packets in ON's buffer at time d_p are $\mathcal{O}$-packets. Because otherwise, ROPT can use an α-value non-$\mathcal{O}$-packet which is only in ON's buffer to replace p ($v_p = 1$) in ROPT's buffer to gain more value. We then claim that there must exist at least one open chain at time d_p since otherwise, each closed chain corresponds to one packet in ROPT's buffer which forbids ROPT to accept p.
   Let q be the earliest-released α-value $\mathcal{O}$-packet in ON's buffer whose corresponding chain is open. In the first step of this chain, say t', ON sends a 1-value non-$\mathcal{O}$-packet (see Corollary 1).
   We charge the value $v_p = 1$ to ROPT in step t' and we close this chain.

Remark 6. For any evicted 1-value $\mathcal{O}$-packet, its value is charged to ROPT in a time step t' in which ON sends a 1-value non-$\mathcal{O}$-packet and t' is the first time step of a closed chain of steps.

Remark 7. For each preempted 1-value $\mathcal{O}$-packet, if its value is charged to ROPT in a time step t' in which ON sends a 1-value non-$\mathcal{O}$-packet and t' is the first time step of a closed chain of steps, the gain ratio in this time step t' is bounded by 1.

Remark 6 and Remark 7 indicate that in the time steps that we charge 1-value evicted/preempted packets to ROPT, the gain ratio is bounded by 1. In the time step when ON sends an $\mathcal{O}$-packet, the value of the $\mathcal{O}$-packet is charged to ROPT in the same time step and the gain ratio is 1. Thus, in order to prove Theorem 1, we only need to analyze the gain ratio for the evicted α-values $\mathcal{O}$-packets and the preempted 1-value $\mathcal{O}$-packets. (Recall that Remark 1 shows that no α-value packet is evicted by ON).

Remark 8. Each evicted (respectively, preempted) $\mathcal{O}$-packet p is associated with a time interval $[d_p,\ l_p - 1]$ (respectively $[d_p,\ d_p + k - 1]$). In the time steps falling in these intervals, ON sends α-value packets only.

To avoid double-charging the $\mathcal{O}$-packets unsent by ON, we have the following results.

Lemma 4. *Consider an interval in which ON sends preempting α-value packets. If there are preempted packets that are charged to ROPT in this interval, then no evicted α-value packets are charged to ROPT in this interval.*

Proof. Note that if a preempted 1-value $\mathcal{O}$-packet p is charged to ROPT in this interval, then at time d_p by when ON preempts p, there are no open chains. From time d_p to the time when ON sends all the preempting packets, no new chains are generated and no open chains exist. Also, for each closed chain, if its last packet is in ON's buffer, the first time step of this chain corresponds to a unique packet in ROPT's buffer. Hence, no α-value packet will be evicted by ON during this interval. □

Corollary 3. *Consider an interval in which ON sends α-value packets. If there are evicted α-value packets charged to ROPT in this interval, then no preempted* 1*-value packets are charged to ROPT in this interval.*

Given Lemma 4 and Corollary 3, to prove Theorem 1, we will show that

1. in each interval ON sends preempting α-value packets, the total value of the preempted 1-value packets assigned to ROPT is bounded by $\frac{1}{\beta}$ times of the value gained by ON; and
2. in each interval ON sends α-value packets, if x evicted α-value $\mathcal{O}$-packets are charged to ROPT in this interval, then that there are x open chains corresponding to these α-value packets. Also, there are x time steps in which ON sends non-$\mathcal{O}$-packets and no values are charged to ROPT in those time steps. Also, x is always bounded by $\frac{\beta}{\alpha+\beta}$.

In the following, we consider the gain ratio for an interval with evicted α-value $\mathcal{O}$-packets.

Lemma 5. *At any time, the number of $\mathcal{O}$-packets which are in ON's buffer but have been sent by ROPT is no more than $\frac{B\cdot\beta}{\alpha+\beta}$.*

Proof. In ON's buffer, the cumulative number of α-value packets that have been sent by ROPT is increased by 1 only when ON sends a 1-value $\mathcal{O}$-packet. That means no preemption happens in that time step; otherwise, that 1-value $\mathcal{O}$-packet will be preempted and an α-value preempting packet will be sent. For each time step that ON sends a 1-value non-$\mathcal{O}$-packet, we have the following inequality (let x be the cumulative number of evicted α-value $\mathcal{O}$-packets): $\alpha < (B-1)\beta$, $2\alpha < (B-2)\beta$, $\cdots$, $x\cdot\alpha < (B-x)\beta$. From $x\cdot\alpha < (B-x)\beta$, we have $x < \frac{B\cdot\beta}{\alpha+\beta}$. □

The above inequality limits the number x of cumulative evicted α-value $\mathcal{O}$-packets that we charge to ROPT in the time interval that ON sends at least B packets with value α. From Lemma 5, $x < \frac{B\cdot\beta}{\alpha+\beta}$. For each evicted α-value packet, it corresponds to a time step in which ON sends a 1-value non-$\mathcal{O}$-packet and in that time step, we do not charge ROPT any value. Thus, in those x time steps and this time interval, ROPT gains a total value of $(x+B')\alpha$ and ON gains a total value of $x + B'\cdot\alpha$, where $B' \geq B$. Then we have the ratio of gains bounded by (note $B' \geq B$ and $x < \frac{B\cdot\beta}{\alpha+\beta}$)

$$\frac{(x+B')\alpha}{x+B'\cdot\alpha} \leq \frac{(x+B)\alpha}{x+B\cdot\alpha} \leq \frac{\left(\frac{B\cdot\beta}{\alpha+\beta}+B\right)\alpha}{\frac{B\cdot\beta}{\alpha+\beta}+B\cdot\alpha} = \frac{\left(\frac{\beta}{\alpha+\beta}+1\right)\alpha}{\frac{\beta}{\alpha+\beta}+\alpha} = \frac{\alpha^2+2\alpha\cdot\beta}{\alpha^2+\alpha\cdot\beta+\beta}.$$

For the interval in which ROPT is charged with preempted 1-value packets, we know that there are no evicted α-value packets are charged to ROPT in this interval (see Lemma 4). Thus, the total value of the preempted 1-value $\mathcal{O}$-packets is bounded by $\frac{1}{\beta}$ times of the total value of the α-value preempting packets. Note that these preempting α-value packets may be $\mathcal{O}$-packets and we charge their values to ROPT in these time steps, thus, the gain ratio is bounded by $\frac{1+\beta}{\beta}$.

We want to minimize the gain ratio ρ for all the time intervals, where $\rho = \max\{\frac{1+\beta}{\beta}, \frac{\alpha^2+2\alpha\cdot\beta}{\alpha^2+\alpha\cdot\beta+\beta}\}$. In order to get $\rho = \frac{1+\beta}{\beta}$ for any α, we have that for any α, $\frac{1+\beta}{\beta} \geq \frac{\alpha^2+2\alpha\cdot\beta}{\alpha^2+\alpha\cdot\beta+\beta}$. This requires

$$\alpha^2 - \beta(\beta-1)\alpha + \beta^2 + \beta > 0. \tag{1}$$

To satisfy the inequality in Equation 1 for any α, we need to guarantee $\beta^2(\beta-1)^2 - 4(\beta^2+\beta) = \beta\left(\beta^3 - 2\beta^2 - 3\beta - 4\right) < 0$. By solving $\beta^3 - 2\beta^2 - 3\beta - 4 < 0$, we have $\beta \leq 3.284$. Hence, we get the gain ratio ρ minimized at 1.304 when $\beta = 3.284$.

Corollary 4. *ON is* 1.304*-competitive when* $\beta = 3.284$.

4 Related Work and Open Problems

Mansour et al. [9] initiated the study of competitive online algorithms for the FIFO buffering model. They designed a simple greedy deterministic algorithm with a tight competitive ratio 2 [4]. The first algorithm with a competitive ratio strictly less than 2 was presented by Kesselman et al. [6]. Englert and

Westermann [3] showed that PG is 1.732-competitive but no better than 1.707-competitive. The lower bound of competitive ratio for deterministic algorithms is 1.409 [6]. For the two-valued variant in which packets have value either 1 or $\alpha > 1$, Kesselman and Mansour [5] proposed a 1.544-competitive memoryless algorithm. Englert and Westermann [3] presented an optimal 1.282-competitive algorithm which meets the lower bound [4]. However, this algorithm [3] is not memoryless.

In this paper, we present a 1.304-competitive memoryless algorithm for the two-valued variant. In [8], an algorithm using marking bits achieves the same competitive ratio 1.304. The algorithm that we present in this paper is simpler and it is not using marking bits. All previous work proactively preempt packets. On the contrary, our algorithm drops packets in a 'lazy' manner. For this variant, closing or shrinking the gaps of [1.282, 1.304] for memoryless algorithms remains an open problem. We hope that the idea presented in this paper will motivate an improved algorithm for the general FIFO model with arbitrary values.

References

1. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
2. Chrobak, M., Jawor, W., Sgall, J., Tichy, T.: Improved online algorithms for buffer management in QoS switches. ACM Transactions on Algorithms 3(4), Article number 50 (2007)
3. Englert, M., Westermann, M.: Lower and upper bounds on FIFO buffer management in QoS switches. Algorithmica 53(4), 523–548 (2009)
4. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. SIAM Journal on Computing (SICOMP) 33(3), 563–583 (2004)
5. Kesselman, A., Mansour, Y.: Loss-bounded analysis for differentiated services. Journal of Algorithms 46(1), 79–95 (2003)
6. Kesselman, A., Mansour, Y., van Stee, R.: Improved competitive guarantees for QoS buffering. Algorithmica 43(1-2), 63–80 (2005)
7. Li, F., Sethuraman, J., Stein, C.: Better online buffer management. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 199–208 (2007)
8. Lotker, Z., Patt-Shamir, B.: Nearly optimal FIFO buffer management for DiffServ. Computer Networks 17(1), 77–89 (2002)
9. Mansour, Y., Patt-Shamir, B., Lapid, O.: Optimal smoothing schedules for real-time streams. Distributed Computing 17(1), 77–89 (2004)

On the Maximum Locally Clustered Subgraph and Some Related Problems

Bang Ye Wu

National Chung Cheng University, ChiaYi, Taiwan 621, R.O.C.
bangye@cs.ccu.edu.tw

Abstract. Motivated by detecting false friend links in online social networks, we define two optimization problems based on the balance theory for structural transitivity in social networks. We give a polynomial time algorithm for one problem and show the NP-hardness of the other. For the NP-hard problem, we show some polynomial time solvable cases and give a 2-approximation algorithm for a restricted version. We also propose a heuristic algorithm for a more general version of the problem.

Keywords: algorithm, social network analysis, time complexity, NP-hard, approximation algorithm.

1 Introduction

A social network is a social structure between actors. The nodes in the network are the actors, while the links show some kind of social ties, such as friendship, co-working or message flow. Social network analysis (SNA) focuses on the analysis of patterns of relationships among actors. Due to the rapid growth of Internet, many social networking platforms are provided and become more and more popular. These platforms provide users to interact with friends and even make friends with others. Nowadays hundreds of million people have registered in many social networking websites, such as Facebook, Twitter, and Plurk [17]. However, due to several reasons, the number of friend-links in such online social networks (OSN) are much more than that in real-life. Today, a user with hundreds or even thousands of friend-links in Facebook is a common phenomenon. One of the reasons is that building a friend-link is much easier than making a friend in real life. Also many people like adding celeblogs as friends (hub-effect). Another reason is that many social networking websites, for example Facebook, provide online games and encourage users to invite others joining the games together. As a result, there are many *false-friend links* in an OSN, and therefore how to identify real-friend links becomes an important issue in SNA.

In this paper, we study the following two graph optimization problems arising in structural analysis of detecting false links. The formal definitions and their reasons are delayed to the next section. An edge is non-triangular if its two endpoints have no common neighbor. The first problem is the *Maximum Constrained Non-triangular Edges* (MCNE) problem, in which we want to find a maximum cardinality edge subset such that, for every node, the number of non-triangular edges incident to it is at most a specified fraction of its degree. The local *clustering coefficient*, or CC in short, of a node in a graph is a basic measurement which quantifies how close the neighborhood of the node is. The second problem in this paper is the *Maximum Locally Clustered Subgraph* (MLCS) problem, in which we want to find a maximum cardinality edge subset such that the CC of every node exceeds a given threshold. Besides modeling the two problems, the contribution of this paper includes the following.

– We show a polynomial time algorithm for the MCNE problem.
– The MLCS problem is NP-hard even for graphs in which all maximal cliques are non-overlapped.
– The MLCS problem can be exactly solved in polynomial time if all non-trivial maximal cliques are disjoint.
– The dual problem of MLCS, i.e., minimizing the number of removed edges, can be 2-approximated in polynomial time if all maximal cliques are non-overlapped and $\theta = 1$.
– We extend the above 2-approximation algorithm to the case of general θ.

Some other related previous works are listed in the follows. The emergency of blogs provides us a opportunity to analyze a huge social network involving millions of people. For example, analyses of several blogs and micro-blogs on social networking websites were reported [1,13,14,7]. All their studies confirm some important features of social networks such as power-law and fat-tail distribution, small-world and local-clustering phenomenon. The correlation between friendship and geographic location in social networks was also studied and found to be strong [10,11], and it is also another reason that a real-life social network is more locally clustered than an online social network. Structure analysis has been used to detect cohesion groups [8,18]. Another approach to identify real-friend links is by *data/text mining*, i.e., analyzing the contents. Some text mining methods are used to find latent friends from blogosphere [2,9,15].

The rest of the paper is organized as follows. In Section 2, we introduce some notations, as well as the reason why the two problems are defined. In Section 3, we show a polynomial time algorithm for the MCNE problem. In Section 4, we discuss the MLCS problem, including the NP-hardness, some polynomial time solvable cases, an approximation algorithm and a heuristic algorithm. Finally a conclusion is given in Section 5.

2 Preliminaries

2.1 Notations and Definitions

An online social network $G = (V, E)$ is a simple undirected graph, in which V and E are the node and edge sets, respectively. An edge (u, v) is a true link if u and v are friends in real-life; or false link otherwise. In this section we define two optimization problems for analyzing true/false links in a social network. We shall use the following notations and terms in graph theory.

For a graph G, $V(G)$ and $E(G)$ denote the node and edge sets, respectively. Two nodes u and v are neighbors of each other if $(u,v) \in E$. The neighborhood of a node N_v is the set of nodes adjacent to v, i.e., $N_v = \{u \in V | (u,v) \in E\}$. Let $N_{uv} = N_u \cap N_v$ denote the co-neighbors of u and v. For a node subset U, the subgraph of G induced by U is denoted by $G[U]$. Also the subgraph induced by an edge set F is denoted by $G[F]$. The degree of a node v in a graph G, denoted by $d(G,v)$ is the number of its neighbors in G. When there is no confusion, we shall simply use $d(v)$. We shall also use $d(F,v)$ to denote $d(G[F],v)$ for short.

A clique is complete subgraph. A k-clique is a clique of k nodes. A k-clique is a *trivial clique* if $k < 3$. A 3-clique is also called as a *triangle*. A clique is *maximal* if it is not properly contained in another clique. Two cliques are *disjoint* if they have no common node, and non-overlapped if they do not share a common edge. An edge (u,v) is a *triangular* edge if it belongs to some triangle; or a *non-triangular* edge otherwise. The sets of all triangular, and non-triangular respectively, edge sets of a graph G are denoted by $T(G)$ and $\bar{T}(G)$, respectively. More formally, $\bar{T}(G) = \{(u,v) \in E(G) | N_{uv} = \emptyset\}$, and $(T(G), \bar{T}(G))$ is a bipartition of E. When there is no confusion, we shall use T and $\bar{T}$ instead of $T(G)$ and $\bar{T}(G)$. By T_v we denote the set of all triangular edges incident to a node v, and $\bar{T}_v$ is defined similarly. In the remaining paragraphs, the input graph is always $G = (V,E)$ and we use $n = |V|$ and $m = |E|$.

The local *clustering coefficient*, or CC in short, of a node v in a graph G is a basic measurement which quantifies how close the neighborhood of node v is [6,16]. The formal definition is given by

$$CC(G,v) = \begin{cases} \frac{2\eta(v)}{d(v)\cdot(d(v)-1)} & \text{if } d(v) \geq 2 \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

in which $\eta(v) = |E(G[N_v])|$ is the number of edges between the neighbors of v. For our convenience we define $CC(v) = 1$ if $d(v) < 2$. In this paper we shall only discuss local CC but not the CC of a group or a entire graph, so we only use "CC" instead of "local CC" in the remaining paragraphs. We shall use $CC(v)$ if we need not specify the graph from the context, and use $CC(F,v) = CC(G[F],v)$ for an edge set F. Similarly we shall use $\eta(F,v)$ if the neighbors are taken from an edge set F, i.e., $\eta(F,v) = |\{(s,t) \in F | (s,v) \in F \wedge (t,v) \in F\}|$.

2.2 Problem Modeling

According to the *balance theory* for structural *transitivity* [3,6,16] in social networks, a real-friend link is usually an edge in some triangle — as in common parlance, "the friend of my friend is also my friend". So, to detect the false links, the first thought is to find all non-triangular edges in the social network. But regarding all non-triangular edges as false links may be overkilled. Since a more active actor usually has more true and false links simultaneously, we define the following optimization problem. The reason of using $d(F,v) + 1$ instead of $d(F,v)$ as the bound is to allow the existence of pending nodes which have no any triangular edge.

Maximum Constrained Non-triangular Edges (MCNE) problem
INSTANCE: A graph $G = (V, E)$ and a positive real $\gamma < 1$.
GOAL: Find a maximum cardinality edge subset $F \subseteq E$ such that, for all nodes $v \in V$, $|\bar{T}_v(G[F])| \leq \gamma \cdot d(F, v) + 1$.

The MCNE problem is only motivated by detecting true links in non-triangular edges. However, there are false links in triangular edges. So it seems too restricted, and we define a more general problem in the following. Also based on the balance theory for structural transitivity, triangles tend to appear frequently in triad pattern connections. As a result, the CC value on the graph formed by the true links should be larger than that on an online social network. Since we aim at finding as many as possible real-friend links, we define the following optimization problem.

Maximum Locally Clustered Subgraph (MLCS) problem
INSTANCE: A graph $G = (V, E)$ and a positive real $\theta \leq 1$.
GOAL: Find a maximum cardinality edge subset $F \subseteq E$ such that, for all nodes $v \in V$, $CC(F, v) \geq \theta$.

The decision version of the problem is to determine if there exists such an edge subset of cardinality larger than or equal to a given bound. The dual MLCS problem is defined similarly except the objective function is defined by the number of removed edges and therefore it is a minimization problem.

3 The MCNE Problem

Recall that, in the MCNE problem, we look for a maximum cardinality edge subset F such that the number of the non-triangular edges incident to v is at most $\gamma d(F, v) + 1$ for every node v. Since the constraint is only on the number of non-triangular edges, any triangle in G should be included in an optimal solution. Furthermore adding non-triangular edges cannot form any triangle. Therefore $|T_v(G[F^*])| = |T_v(G)|$ for any optimal solution F^*. For any feasible solution F, we have

$$|\bar{T}_v(G[F])| \leq \gamma \cdot d(F, v) + 1 = \gamma(|\bar{T}_v(G[F])| + |T_v(G[F])|) + 1$$
$$|\bar{T}_v(G[F])| \leq \frac{1}{1-\gamma}(\gamma |T_v(G[F])| + 1) \quad (2)$$

After finding $T(G)$, we can easily determine the upper bound, named $b(v)$, of the number of non-triangular edges incident to every node v by Eq. (2). The MCNE problem can be reduced to find a maximum subgraph of $G[\bar{T}]$ such that the degree of any node v is upper bounded by $b(v)$. This problem is known by the name *upper degree-constrained subgraph* (UDCS) [4] or, more generalized, the *b-matching* problem. The UDCS problem is a generalization of graph matching problem ($b(v) = 1$ for all nodes v) and can be solved by reducing to the maximum matching problem. The algorithm for solving the MCNE problem is as follows.

Algorithm 1

```
Input: a graph $G = (V, E)$ and a positive real $\gamma < 1$;
Output: an optimal solution $F$ of the MCNE problem;
1: find $T(G)$ and $\bar{T}(G)$;
2: compute $b(v)$ by Eq. (2) for each node $v$;
3: find an edge subset of $F$ by solving a UDCS problem with input $G[\bar{T}]$ and $b$;
4: output $T(G) \cup F$.
```

Lemma 1. *$T(G)$ and $\bar{T}(G)$ can be computed in $O(mn)$ time.*

Proof. To find $T(G)$ is equivalent to determining if $N_{uv} = \emptyset$ for all edges $(u, v) \in E$. If the graph is stored in an adjacency matrix, it can be easily done in $O(n^3)$ time. For a sparse graph stored in an adjacency list, determining if $N_{uv} = \emptyset$ can be done in time linear to $d(u) + d(v)$. Therefore the total time complexity is $\sum_{(u,v)\in E}(d(u) + d(v)) = 2\sum_{(u,v)\in E} d(v) = 2\sum_{v\in V} d(v)^2 \in O(mn)$ since $\sum d(v) = 2m$ and $d(v) \leq n - 1$ for any $v \in V$. □

By [4], the UDCS problem can be solved in $O((\sum_v b(v))^{1/2}m)$. In this application $b(v) < n$, and therefore $(\sum_v b(v))^{1/2} < n$. Together with Lemma 1, we obtain the next theorem.

Theorem 1. *The MCNE problem can be solved in $O(mn)$ time.*

4 The MLCS Problem

We discuss the MLCS problem in this section. We show the NP-hardness in Section 4.1. Then we give some polynomial time algorithms for some special cases in Section 4.2. In Section 4.3, we first give a 2-approximation algorithm for the non-overlapped case and $\theta = 1$, and then extend it to a heuristic algorithm for general θ.

4.1 The NP-Hardness

In this subsection we show the next theorem.

Theorem 2. *The MLCS problem is NP-hard even when all maximal cliques are non-overlapped.*

We first show the NP-hardness of a restricted version of the *partition into triangles* (PIT) problem, and then the theorem is shown by transforming from the restricted PIT problem. The restricted PIT problem is as follows.

> Restricted Partition Into Triangles (Restricted PIT) problem
> INSTANCE: A graph $G = (V, E)$ with $|V| = 3q$ for some integer q, in which any two triangles in G are non-overlapped.
> QUESTION: Can the nodes of G be partitioned into q disjoint sets $V_1, V_2, ..., V_k$ such that the subgraph induced by each V_i is a triangle?

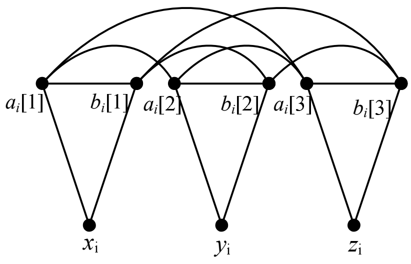


Fig. 1. Local replacement for $c_i = (x_i, y_i, z_i)$ for transforming X3C to restricted PIT

Note that "all triangles are non-overlapped" also implies there is no 4-clique. The PIT problem was shown to be NP-complete by transforming from the *Exact 3-cover* (X3C) problem [5, p. 68]. In an X3C problem, we are given a collection of 3-element subsets of a universal set, and we want to determine if there exists a sub-collection such that every element occurs exactly once in the sub-collection. The only difference of the restricted PIT defined above from the previous one is that we require the triangles are non-overlapped. Unfortunately, the reduction in [5] consists of overlapped triangles and cannot be used directly. Instead we design another local replacement for a subset $c_i = (x_i, y_i, z_i)$ in the X3C problem as in Figure 1.

The local replacement consists of two *internal* triangles $(a_i[1], a_i[2], a_i[3])$ and $(b_i[1], b_i[2], b_i[3])$, as well as 3 *external* triangles $(x_i, a_i[1], b_i[1])$, $(y_i, a_i[2], b_i[2])$ and $(z_i, a_i[3], b_i[3])$. First we can see that the triangles are not overlapped and there is no 4-clique. If the graph can be partitioned into triangles, the three external triangles are either all or none in the partition. The remaining of the proof is the same as the previous one, and we obtain the next lemma.

Lemma 2. *The restricted PIT problem is NP-complete.*

We prove Theorem 2 by transforming the restricted PIC to the MLCS with threshold $\theta = 1$. Supposed $G = (V, E)$ is an instance of the restricted PIT problem, in which all triangles are non-overlapped. The corresponding instance of the MLCS problem is G and $\theta = 1$. For any node v, $CC(v) = 1$ iff its neighborhood forms a clique or $d(v) < 2$. It is trivial that the optimal solution of the MLCS problem contains $3q$ edges if the graph can be partitioned into triangles. Conversely suppose that there exists a subgraph with $3q$ edges and $CC(v) = 1$ for each node v. Since there is no clique of 4 nodes, the total number of edges in the subgraph is at most $3q$ and the equality holds when each maximal clique contains 3 nodes. Therefore G can be partitioned into triangles.

4.2 Polynomial Time Solvable Cases

The following theorem shows a special case for which the MLCS problem can be easily solved. Note that it does not matter what value θ is.

Theorem 3. *When G contains no 3-clique, the optimal solution of the MLCS problem is a maximum cardinality matching of G and therefore can be solved in $O(\sqrt{n}m)$ time.*

Proof. Since there is no 3-clique, for any node in any subgraph of G, its CC is either 0 or 1. Therefore any feasible solution can only consist of disjoint edges and isolated nodes, i.e., a matching of G. The result follows from that a maximum cardinality matching is the one with as many edges as possible. The time complexity comes from [12]. □

Next we aim at generalizing the above theorem to the case that all non-trivial maximal cliques are disjoint. We first show some properties. Suppose that the current degree of a node v is d_v and η_v is the number of edges between these neighbors. Consider $f_v(k, \eta_v)$ as the $CC(v)$ when we attach an additional $(k+1)$-clique to v for $k \geq 1$. Note that f_v is defined only for $k \geq 1$.

$$f_v(k, \eta_v) = \frac{\eta_v + \binom{k}{2}}{\binom{d_v+k}{2}} = \frac{2q + k(k-1)}{(d_v + k)(d_v + k - 1)} \tag{3}$$

Lemma 3. *$f_v(k+1, \eta_v) \geq f_v(k, \eta_v)$ when $k \geq 2\eta_v/d_v$.*

Proof. By Eq. (3), $f_v(k+1, \eta_v)/f_v(k, \eta_v) \geq 1$ iff

$$\left(\frac{2\eta_v + k(k+1)}{2\eta_v + k(k-1)}\right)\left(\frac{d_v + k - 1}{d_v + k + 1}\right) \geq 1 \Leftrightarrow k \geq \frac{2\eta_v}{d_v}$$

□

The next corollary immediately follows from the above lemma.

Corollary 1. *$f_v(k, 0)$ is increasing as k increases.*

Lemma 3 also implies that $f_v(k, \eta_v)$ is *bitonic*, i.e., it first decreases and then increases. For a preexisting edge set F and a node v, the size of a clique which can be attached to v without violating the requirement of CC can be determined by its current degree and η_v, and importantly the range of feasible sizes has the form as shown in the next corollary.

Corollary 2. *The range of k such that a k-clique can be attached to a node to satisfy the CC requirement has the form $[0, l] \cup [u, \infty]$, in which $l \leq u$ are determined by η_v and d_v.*

Next, the number $b(v)$ of non-triangular edges incident to a node v, which can be added without violating the requirement of CC, can be easily computed by

$$\frac{2\eta_v}{(d_v + b(v))(d_v + b(v) - 1)} \geq \theta \tag{4}$$

Lemma 4. *If all maximal non-trivial cliques are disjoint, there exists an optimal solution of the MLCS problem containing all the non-trivial cliques.*

Proof. Suppose that F is an optimal edge set and C is a non-trivial maximal clique such that $E(C) - F \neq \emptyset$. We construct an edge set F' from F as follows. First let $F' = F \cup E(C)$. We consider any node $v \in V(C)$. Let $p_1 = d(F \cap E(C), v)$ and $p_2 = d(F - E(C), v)$. We only need to consider those nodes which have more neighbors in F' than in F, i.e., $p_1 < |V(C)| - 1$. Since all non-trivial cliques are disjoint, the edges incident to v but not in C are all non-triangular edges. The case $p_1 = 1$ and $p_1 > 0$ is impossible because $CC(v) = 0$ in such a case. If $p_1 > 1$, by Corollary 1, $CC(F', v) \geq CC(F, v)$. The remaining case is that $p_1 = 0$. When $p_1 = 0$, $p_2 \leq 1$ or otherwise $CC(F, v) = 0$. If $p_2 = 0$, we have $CC(F', v) = 1 \geq \theta$. If $p_2 = 1$, we remove the edge previously incident to v. The removal decreases the total degree by two but $d(F', v) \geq d(F, v) + 2$, and therefore the total number of edges is not decreased. Also, since all non-trivial cliques are disjoint, the removed edge must be a non-triangular, and its removal does not decrease the CC of any node. □

Algorithm 2

```
Input: a graph $G$ with disjoint non-trivial maximal cliques and a real $0 < \theta \leq 1$;
Output: An optimal solution of the MLCS problem;
1: $S \leftarrow \emptyset$;
2: find all non-trivial maximal cliques and put them into $S$;
3: let $F$ be the set of all non-triangular edges in $G$;
4: compute $b(v)$ by Eq. (4) for each node $v$;    ▷ maximum number of non-triangular
   edges which can be incident to $v$
5: find an edge subset of $F$ by solving a UDCS problem with $G[\bar{T}]$ and $b$;
6: add $F$ into $S$;
7: return $S$.
```

Theorem 4. *If all non-trivial maximal cliques are disjoint, the MLCS problem can be solved by Algorithm 2 in $O(mn)$ time.*

Proof. By Lemma 4, there exists an optimal solution containing all non-trivial maximal cliques. The remaining edges are all non-triangular edges. Similar to solving the MCNE problem, the maximum number of non-triangular edges in such an optimal solution can be found by solving a UDCS problem.

Since all non-trivial maximal cliques are disjoint, similar to Lemma 1, they can be found in $O(mn)$ time. Computing $b(v)$ for every v takes only linear time. Finally, by [4], the UDCS problem can be solved in $O(mn)$ time since $\sum_v b(v) < n^2$. □

4.3 Algorithms for Graphs with Non-overlapped Maximal Cliques

We focus on approximation and heuristic algorithms for the MLCS problem with non-overlapped maximal cliques. We name the restricted version as the MLCSN problem, and the dual MLCSN problem is defined similarly except the measurement is the number of removed edges. When θ is fixed to 1, the problem is called by MLCSN$_1$. First we give a 2-approximation algorithm for the dual MLCSN$_1$ problem.

Algorithm 3. A 2-approximation algorithm for the dual MLCS problem with non-overlapped maximal cliques and $\theta = 1$

```
1: $S \leftarrow \emptyset$;
2: find all non-trivial maximal cliques and store them in a list $L$;
3: while cliques in $L$ are not disjoint do
4:     extract a clique $C$ with maximum cardinality from $L$;
5:     $S \leftarrow S \cup E(C)$;
6:     for all clique $Q$ in $L$ do
7:         $V(Q) \leftarrow V(Q) - V(C)$;
8:         if $|V(Q)| < 3$ then remove $Q$ from $L$;
9:     end for
10: end while
11: let $V'$ be the set of nodes not in $S$;
12: find an optimal solution $M$ of $G[V']$ by Algorithm 2;
13: return $S \cup M$.
```

To simplify the performance analysis, we analyze the simpler version of Algorithm 3, in which we put all maximal cliques into L (including maximal 2-cliques) initially and the while-loop is continued until all cliques in L are singleton (the while-condition and the if-condition at Step 8 are modified). Algorithm 3 performs better than the simpler version because their behaviors are the same before exiting the while-loop, and after that, Algorithm 3 finds an optimal solution of the remaining graph.

Let F^* be an optimal solution for the MLCSN$_1$ problem. For any node v, let $\hat{C}_v$ denote the maximum clique containing v and $c_v = |V(\hat{C}_v)| - 1$, i.e., the number of neighbors of v in $\hat{C}_v$.

Lemma 5. $|E - F^*| \geq (1/2)\sum_{v \in V}(d(v) - c_v)$.

Proof. When $\theta = 1$, any feasible solution is a set of disjoint cliques. By the definition of c_v, for any node $v \in V$, $d(F^*, v) \leq c_v$. That is, the minimum degree decrement of v is at least $(d(v) - c_v)$ and therefore the total degree decrement is at least $\sum_{v \in V}(d(v) - c_v)$. □

Lemma 6. *Algorithm 3 drops at most* $\sum_{v \in V}(d(v) - c_v)$ *edges.*

Proof. The edges are dropped at Step 7 when we put a clique C to S and remove $V(C)$ from any other clique joint to C. Consider any node $v \in V(C)$. Some neighbors of v may have been removed before this step. Let $d'(v)$ be the current degree of v and c'_v the number of neighbors of v in $\hat{C}_v$ which have not been put into S yet. Since C is a largest remaining clique, we have $|V(C)| - 1 \geq c'_v$. The number of edges incident to v and removed at this iteration is

$$d'(v) - (|V(C)| - 1) \leq d'(v) - c'_v \leq d(v) - c_v.$$

Summing over all nodes in C and all iterations, the result follows. □

Theorem 5. *Algorithm 3 is a 2-approximation algorithm for the dual MLCSN$_1$ problem and runs in $O(mn)$ time.*

Proof. The approximation ratio follows from Lemmas 5 and 6. Since all maximal cliques are non-overlapped, the maximal cliques can be found by a simple incremental method. Similar to Lemma 1, they can be found in $O(mn)$ time. Also the total number of edges of the maximal cliques is at most m. Each iteration of the while-loop can be done in $O(m)$ and the number of iterations is $O(n)$ because at least two nodes are removed in one iteration. Finally, by Theorem 4, Algorithm 2 runs in $O(mn)$ time, and therefore the total time complexity is $O(mn)$. □

Next we extend Algorithm 3 to a heuristic algorithm for the MLCSN problem with general θ. The algorithm is list below.

Algorithm 4. A heuristic algorithm for the MLCSN problem

```
 1: $S \leftarrow \emptyset$;
 2: find all non-trivial maximal cliques and store them in a list $L$;
 3: loop
 4:     extract a clique $C$ with maximum cardinality from $L$;
 5:     if $|V(C)| < 3$ then
 6:         exit the loop;
 7:     end if
 8:     find a maximum sub-clique $C'$ of $C$ such that $CC(S \cup E(C'), v) \geq \theta$ for every
        $v \in V(C')$;
 9:     if $|E(C')| \geq 3$ then
10:         $S \leftarrow S \cup E(C')$;
11:     end if
12:     insert $C - C'$ into $L$;
13: end loop
14: let $F$ be the set of edges not in $S$;
15: compute $b(v)$ as the maximum number of non-triangular edges which can be incident to $v$ for each node $v$;
16: find an edge subset $S'$ by solving a UDCS problem with input $G[F]$ and $b$;
17: $S \leftarrow S \cup S'$;
18: return $S$.
```

Similar to the algorithm for $\theta = 1$, we always keep S a feasible solution and try to add a clique as large as possible into the solution. Different from the case $\theta = 1$, a node may be included in more than one cliques. Therefore we should not remove a node from the clique list as in the previous algorithm. When the currently largest clique is considered, the nodes of the clique may have been attached to different preexisting cliques. So it is possible that only partial but not entire clique can be added to the solution. It is still unknown if the algorithm ensures a performance ratio. We shall aim at how to implement Step 8.

The situation at Step 8 is that there is a preexisting edge subset S and a clique C such that $S \cap E(C) = \emptyset$ (by the assumption that non-trivial cliques are non-overlapped). We want to find a largest sub-clique C' of C such that, for each $v \in V(C')$, attaching C' to v does not violate the requirement of CC, i.e.,

$$\eta(S,v) + \binom{|V(C')|-1}{2} \geq \theta \cdot \binom{d(S,v)+|V(C')|-1}{2}$$

By Corollary 2, there exist two integers $b_1(v) \leq b_2(v)$ for each node v such that the size of a clique which can be attached to v without violating the CC requirement is in the range $[0, b_1(v)] \cup [b_2(v), \infty]$. It is possible that $b_1(v) = 0$ or $b_2(v) \geq |V(C)|$, and the range degenerates into one interval or even $\emptyset$. For the latter case we can simply ignore the node. Anyway, the problem can be simplified as follows:

> Given two integers $b_1(i)$ and $b_2(i)$ satisfying $0 \leq b_1(i) \leq b_2(i)$ for i from 1 to k, find a maximum cardinality subset R of $\{1..k\}$ such that, for each $i \in R$, $|R| \geq b_2(v)$ or $|R| \leq b_1(v)$.

We shall call it the *Maximum Subset with Range Constraint* (MSRC) problem. A naive algorithm for this problem works as follows. For i from k down to one, check if there are i elements satisfying the range constraint. The time complexity is obviously $O(k^2)$. The problem can also be thought of as follows. Given at most $2k$ intervals $[l_i, r_i]$ in which the endpoints are all integers between 0 and k, we want to know, for each integer i between 1 and $k-1$, the number of intervals containing i. The following simple algorithm works in linear time. First we sort the endpoints and then scan these endpoints from small to large. When a right endpoint is encountered, the number of intervals is decreased by one; and when a left endpoint is encountered, the number of intervals is increased by one. The scan process can be done in linear time since there are at most $4k$ endpoints. Besides, since all endpoints are integers in the range $[0, k]$, the sorting process also takes only linear time. We have the next lemma.

Lemma 7. *The MSRC problem can be solved in linear time.*

Lemma 8. *The time complexity of Algorithm 4 is $O(mn)$.*

Proof. Similar to Theorem 5, the total edge number of all cliques are linear to m, and all the maximal cliques can be found in $O(mn)$ time. At each iteration of the loop, we extract a maximum clique from L, solve an MSRC problem, then delete some node from the clique, and put the remainder back into L. Using a priority queue to implement L, the total time complexity for the extracting step is $O(m \log m)$ since total edge number of all cliques in L is decreased by at least three at each iteration. Suppose that initially the number of nodes of the maximal cliques are k_i, $1 \leq i \leq c$. An initial k_i-clique may consume up to $O(k_i^2)$ time since in worst case its node number is decreased by three each time chosen from L and solving the corresponding MSRC problem takes time linear to its node number. Therefore The total time complexity of the loop is

$O(m \log m + (\sum_i k_i^2))$. Since the total edge number of all cliques are linear to m, that is, $\sum_i \binom{k_i}{2} \in O(m)$, we have that $\sum_i k_i^2$ is also $O(m)$. Finally, similar to Theorem 4, the UDCS problem can be solved in $O(mn)$ time, and the time complexity of the whole algorithm is therefore $O(mn)$. $\square$

5 Conclusion

In this paper, we define two optimization problems arising in detecting false friend links in online social networks. The modeling is based on the balance theory for structural transitivity in social networks. We study the two problems in the algorithmic aspect. We design a polynomial time algorithm for the MCNE problem but the MLCS problem is shown to be NP-hard. For the MLCS problem, the polynomial time algorithm and the 2-approximation algorithm proposed in this paper are only for some restricted cases which rarely happen in a real online social network. However, they may lead to some heuristic algorithms for the general case. We conclude this paper by listing several open problems.

First, there are several theoretically interesting questions about the MLCS problem: its approximabilities for the disjoint-cliques case, for the non-overlapped-cliques case, and for the general case. The performance of the heuristic algorithm proposed in this paper has not been tested, and heuristic algorithms for the general case are expected. In the practical view, the effectiveness of the modeled problems should be further investigated, by simulated data and real data.

Acknowledgment. This work was supported in part by NSC 97-2221-E-194-064-MY3 and NSC 98-2221-E-194-027-MY3 from the National Science Council, Taiwan.

References

1. Ahn, Y.Y., Han, S., Kwak, H., Jeong, S.H.: Analysis of topological characteristics of huge online social networking services. In: Proc. of the 16th international conference on World Wide Web, pp. 835–844 (2007)
2. Chin, A., Chignell, M.: Finding evidence of community from blogging co-citations: a social network analytic approach. In: Proc. of 3rd IADIS International Conference Web Based Communities 2006 (WBC 2006), San Sebastian, Spain, pp. 191–200 (2006)
3. Davis, J.A.: Structural balance, mechanical solidarity, and interpersonal relations. American Journal of Sociology 68, 444–462; Electronics and Communications in Japan (Part I: Communications) 89(12), 88 – 96 (1963)
4. Gabow, H.N.: An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In: STOC, pp. 448–456 (1983)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to The Theory of NP-Completeness. Freeman, NewYork (1979)
6. Hanneman, R.A., Riddle, M.: Introduction to Social Network Methods (2005), http://www.faculty.ucr.edu/hanneman/nettext/

7. Java, A., Song, X., Finin, T., Tseng, B.: Why we twitter: An analysis of a microblogging community. In: Zhang, H., Spiliopoulou, M., Mobasher, B., Giles, C.L., McCallum, A., Nasraoui, O., Srivastava, J., Yen, J. (eds.) WebKDD 2007. LNCS, vol. 5439, pp. 118–138. Springer, Heidelberg (2009)
8. Kuan, S.T., Wu, B.Y., Lee, W.J.: Finding friend groups in Blogsphere. In: Proc. of the 22nd International Conference on Advanced Information Networking and Applications, pp. 1046–1050 (2008)
9. Li, Q., Xu, M., Hou, J., Liu, F.: Web classification based on latent semantic indexing. Journal of Communication and Computer 3(1), 24–27 (2006)
10. Liben-Nowell, D., Novak, J., Kumar, R., Raghavan, P., Tomkins, A.: Geographic Routing in Social Networks. Proc. of the National Academy of Sciences (PNAS) 102(33), 11623–11628 (2005)
11. Lin, J., Halavais, A., Zhang, B.: Blog network in America: blogs as indicators of relationships among U.S. cities. Connections 27(2), 15–23 (2007)
12. Micali, S., Vazirani, V.V.: An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In: FOCS, pp. 17–27 (1980)
13. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proc. of the 5th ACM/USENIX Internet Measurement Conference (IMC 2007), San Diego, CA (2007)
14. Mislove, A., Koppula, H.S., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Growth of the flickr social network. In: Proc. of WOSN, Seattle, WA (2008)
15. Shen, D., Sun, J.T., Yang, Q., Chen, Z.: Latent friend mining from blog data. In: Sixth IEEE International Conference on Data Mining (ICDM 2006), pp. 552–561 (2006)
16. Wasserman, S., Faust, K.: Social Network Analysis. Cambridge University Press, Cambridge (1994)
17. Wikipedia, `http://en.wikipedia.org/wiki/`.
18. Yang, C.-P., Liu, C.-Y., Wu, B.Y.: Influence clubs in social networks. In: Pan, J.-S., Chen, S.-M., Nguyen, N.T. (eds.) ICCCI 2010. LNCS, LNAI vol. 6422, pp. 1–10. Springer, Heidelberg (2010)

Quickest Paths in Anisotropic Media

Radwa El Shawi[1,2] and Joachim Gudmundsson[1,2]

[1] School of Information Technology, University of Sydney
[2] NICTA, Sydney, Australia
radwa.elshawi@nicta.com.au, joachim.gudmundsson@sydney.edu.au

Abstract. In this paper we study the quickest path problem where speed is direction-dependent (anisotropic). The problem arises in sailing, robotics, aircraft navigation, and routing of autonomous vehicles, where the speed is affected by the direction of waves, winds or slope of the terrain. We present an approximation algorithm to find a quickest path for a point robot moving in planar subdivision, where each face is assigned a translational flow that reflects the cost of travelling within this face.

Our main contribution is a data structure that given a subdivision with translational flows returns a $(1+\varepsilon)$-approximate quickest path in the subdivision between any two query points in the plane.

1 Introduction

Geometric shortest path problem is one of the fundamental problems studied in computational geometry and other areas including graph algorithms, geographical information systems (GIS) and robotics. An important category of this problem is to determine the quickest path between a source point s and a destination point t in a geometric environment. In many cases the environment is modelled as a triangular subdivision. Different metrics may be used in all cells of the subdivision to represent some additional mechanical constraints such as friction, flow or steepness. The mechanical constraint we consider in this paper is the flow which can, for example, be an air flow or a water flow.

We assume that the problem (we will use the same notations and definitions as in [14]) is given as a planar triangular subdivision where each face r defined by the subdivision is assigned a *translational flow* defined by a vector $\overrightarrow{f_r}$. Each region r is also assigned a non-negative real number b_r giving the maximum Euclidean norm of the control velocity that the robot can apply within r. We define ρ_r to be the ratio between b_r and $|\overrightarrow{f_r}|$.

Assume we are considering the movement of a robot. The robot is considered to be a point with a given initial position and also a given final position. At time $\tau = 0$, the point robot is at the given initial position point. Within each region r, the robot can apply, at each time $\tau = 0$ and in any direction, a *translational control velocity vector* $\overrightarrow{v(\tau)}$ of bounded Euclidean norm $|\overrightarrow{v(\tau)}| = b_r$. However, the actual velocity of the robot at time τ is given by the sum of its control velocity vector $\overrightarrow{v(\tau)}$ and the translational flow velocity $\overrightarrow{f_r}$ of region r, see Fig. 1.

The *flow path optimization problem* is to find an optimal path of movement of the robot from the initial to the final position of minimum time duration. This problem has a wide range of applications (see also [14]), for example:

– Navigating a vessel on the ocean through regions with different currents. In particular finding a path with minimum fuel consumption from a source point to a destination point.
– Finding a quickest path on a terrain where going up or down affects the maximum speed.
– Finding a cheapest path (in terms of fuel consumption) for an aircraft moving through regions with different wind conditions.

The direction-dependent structure of the problem results in an asymmetric cost function, where the cost of traversing a straight line segment ab, is not necessarily equal to that of a reversed link ba. Thus, the cost function is not a metric, consequently restricting the set of mathematical tools available to us.

1.1 Problem Formulation

In the following we review the algorithm by Reif and Sun [14] to compute an optimal path, as our approach builds upon their work. For any two points p, $q \in \mathbb{R}^2$, we denote by $\overline{pq}$ the closed, oriented line segment from p to q. We denote by $|\overline{pq}|$ the Euclidean distance between p and q. We will show how to compute an optimal path from a source point u to a destination point u' inside a region r. Reif and Sun [14] showed that an optimal path is simple, piecewise linear and it only changes direction between regions.

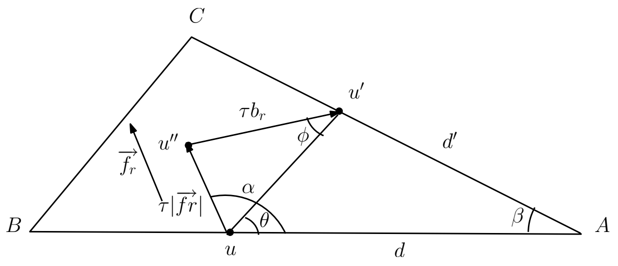


Fig. 1. Computing $\tau(u, u')$

Let $r = \triangle ABC$ be a region with flow $\overrightarrow{f_r}$ and let $\beta = \angle BAC$, as shown in Fig. 1. Let u be a point on $\overline{AB}$ with distance d to A and let u' be a point on $\overline{AC}$ with distance d' to A. Let α be the angle between $\overline{BA}$ and $\overrightarrow{f_r}$ and let θ be the angle $\angle u'uA$. For a robot to travel from a source point u to a destination point u' with minimum time, it needs to apply a control velocity $\overrightarrow{v_r}$ with magnitude b_r. Referring to Fig. 1 we can draw a "virtual triangle" $\triangle uu'u''$ such that $\angle u''uu' = \alpha - \theta$, $\overline{uu''} = \tau(u, u') \cdot |\overrightarrow{f_r}|$ and $\overline{u''u'} = \tau(u, u') \cdot b_r$. In the virtual triangle $\triangle uu'u''$, the vector $\overline{uu''}$ represents the transactional flow velocity, $\overline{u''u'}$ represents the control velocity of a robot in region r and $\overline{uu'}$ represents the composite vector of the robot movement. The following lemma (refer to Fig. 1) shows how an optimal path is computed.

Lemma 1 (Adapted from Lemma 1 in [14]). *The face-wise optimal path from u to u' can be achieved by adopting a control velocity with maximum magnitude b_r and an angle of $\phi = \arcsin(\frac{\sin(\alpha-\theta)}{\rho_r})$ from $\overline{uu'}$. Further, the cost $\tau(u, u')$ of this path is $\frac{l}{b_r(\cos\phi+\cos(\alpha-\theta)/\rho_r)}$, where $l = |\overline{uu'}| = \sqrt{d^2 + d'^2 - 2dd'\cos\beta}$.*

1.2 Previous Work

Optimal path planning problems have been studied for a very long time. In the following we discuss the most related work and refer the interested reader to the survey by Mitchell [10].

In the weighted subdivision problem [11], a point robot moves within a planar subdivision, each face f of the subdivision is assigned a weight $w > 0$. The cost of a path within a face f is the length of this path multiplied by w. Let n denote the number of vertices of the triangular subdivision. The first approximation algorithm for the weighted subdivision problem was by Mitchell and Papadimitriou [11]. Their algorithm used a continuous Dijkstra method to find an optimal path between any source point and destination point. The complexity of their algorithm is $O(n^8 M)$, where M is a function in many parameters including $\varepsilon > 0$, a parameter specifying the degree of the precision. Mata and Mitchell [9] presented a $(1+\varepsilon)$ approximation algorithm. The algorithm is based on constructing a relatively sparse graph, a "pathnet", that links selected pairs of subdivision vertices (and "critical points of entry") with locally optimal paths. The running time of this algorithm is $O(\frac{n^3 N^2 w_{max}}{\varepsilon w_{min}})$, where N is the maximum coordinate of the vertices and $w_{max}(w_{min})$ is the maximum (minimum) weight of a triangular region.

Other algorithms for the weighted subdivision problem discretize the polygonal subdivision by placing Steiner points along the edges of the subdivision and then finding a quickest path in a graph whose nodes are Steiner points or vertices of the subdivision and whose edges are line segments. Then an optimal path is computed on the resulting graph. In particular, Lanthier et al. [8] presented a $(1+\varepsilon)$-approximation algorithm based on uniform discretization for the polygonal regions which adds $m = O(n^2)$ points on each edge and then they constructed a spanner graph in which they computed the approximate path. The time complexity of this algorithm is $O(\frac{n^3}{\varepsilon}\log n)$, as the graph has $O(n^3)$ vertices and $O(n^3/\varepsilon)$ edges. Aleksandrov et al. [1] proposed a logarithmic discretization scheme. The running time of their algorithm is $O(kn\log kn)$, where $k = O(\log_t(L/r'))$ and L denotes the length of the longest edge in the subdivision. Here r' is used to represent the minimum distance from any point to the boundary of the regions adjacent to it and $t = 1 + \frac{\varepsilon w_{max}}{w_{min}\sin(\theta_{min})}$, where θ_{min} is the minimum angle in the subdivision.

The main limitation of the weighted subdivision model is that it only models situations where metrics are isotropic. It can not model the effect of current, wind or any other types of forces. Only a handful of papers looked at the problem of finding the quickest path in anisotropic media. Papadakis and Perakis [13] gave heuristic algorithms for related problems such as minimal time vessel routing among ocean currents. Rowe [15] discussed optimal path planning for a mobile robot with direction dependent forces (friction and gravity).

Reif and Sun [14] gave an approximation algorithm for the motion planning in the presence of uniform flows. The anisotropy was introduced as a uniform flow assigned to each region. Then, the actual velocity of an object is defined to be the sum of a flow vector and a control velocity. The complexity of their algorithm is $O(\frac{n \cdot c_{skew}}{\varepsilon}(\frac{c_{skew}}{\varepsilon} \log \frac{c_{skew}}{\varepsilon} + \log n) \log \frac{c_{skew}}{\varepsilon})$, where C_{skew} is defined as follows. Let $\lambda = \max\{c_f/c_{f'}$: adjacent faces f and $f'\}$, where c_f and $c_{f'}$ are the maximum control velocities applied in regions f and f' respectively. Then $C_{skew} = \Theta(\frac{\lambda(w_{min}+1)}{\theta_{min}(w_{min}-1)})$.

Cheng et al. [5] generalized the problem studied by Reif and Sun [14]. Nevertheless, they limited their research to the case where the speed function has a very specific structure. Each face of the subdivision is assigned a convex distance function that has the following property: its unit disk contains a unit Euclidean disk, and is contained in a Euclidean disk with radius ρ. The running time of their algorithm is $O((\rho^2 \log \rho/\varepsilon^2)n^3 \log(\rho n/\varepsilon))$.

1.3 Our Contribution

In this paper we make a small modification to the algorithm by Reif and Sun [14] that improves the running time of their algorithm by roughly a factor of $\frac{c_{skew}}{\varepsilon}$. However, the main contribution of the paper is a data structure for the query version of the problem.

The proposed algorithm constructs a graph and then finds an approximate quickest path in this graph whose nodes are Steiner points or vertices of the subdivision and whose edges are line segments connecting the points. Reif and Sun [14] construct this graph by connecting every pair of points in the same region. In order to reduce the complexity of the graph, we use the well-separated pair decomposition (WSPD) [1] which allows us to only use a subgraph of the complete graph in each region. Then an approximate path can be obtained from the graph. Such a path approximates an optimal path in the original continuous space.

The query version of the problem uses the above graph together with ideas from the construction of so-called θ-graphs to answer an approximate quickest path query in logarithmic time.

This paper is organized as follows. Next we present a $(1 + \varepsilon)$-approximation algorithm for the basic optimisation version of the problem. Then, in Section 3, we consider the query version. That is, preprocess the input such that an approximate quickest path between two query points s and t can be answered efficiently. We conclude with some remarks and open problems in Section 4.

2 An Efficient $(1 + \varepsilon)$ Approximation Algorithm

We consider a simple improvement of Reif and Sun's [14] construction. In their paper they transformed the geometric continuous problem into a discrete combinatorial problem (graph problem). This graph was built by placing Steiner points on the boundary of the triangular regions. For every region a subgraph is built by connecting all possible pairs of Steiner points and original vertices of the region. This also includes the source and end points which are connected to all points in their respective regions. Then all the subgraphs are combined into one graph. On the resulting graph, a quickest path is computed using Dijkstra's algorithm. The path was proven, in [14], to be an approximate shortest path.

Instead of using a complete graph in each region, as in [14], we construct a graph $G'(V, E')$ by only adding a linear number of edges in each region using the WSPD which results in a reduction of the graph size and hence an improvement of the running time. We show that there exists a path between s and t in G' with cost $(1 + \gamma)$ times the cost of an optimal path in graph G, where γ is a positive constant given as part of the input. The running time of our algorithm is the time for constructing the graph plus the time for running a shortest path algorithm on the graph G'.

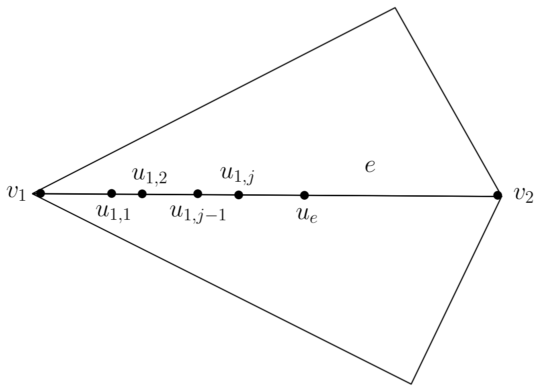


Fig. 2. Adding Steiner points on a boundary edge

2.1 Placing Steiner Points

To compute a γ-good path i.e., a path whose cost is at most $(1 + \gamma)$ times the cost of an optimal path, we use the same logarithmic discretization schema used by Reif and Sun [14]. Let $\tau_{\min}(u)$ denote the minimum time of travelling along a straight line path between u on a boundary edge and any point on a boundary edge not incident to u. For any boundary edge e, let u_e denote the point on the boundary edge e with maximum $\tau_{\min}(u)$.

This discretization scheme places a higher number of Steiner points in the portions of e closer to the endpoints. At the same time the scheme places a smaller number of Steiner points in portions closer to u_e. The intuition is that, roughly speaking, if an optimal segment (a segment of an optimal path) crosses an edge e at a point close to u_e, the segment will be relatively long. Therefore, it is always possible to find an approximate segment (a segment connecting two Steiner points) that neighbors the optimal segment. Further, the cost of this approximate segment is not more than $(1 + \gamma)$ times the cost of an optimal segment.

In the following we will show how the Steiner points are placed on boundary edges. Let b_e be the lesser of the maximum composite velocities of a robot travelling in either direction on e. For any vertex v, we use R_v to give a lower-bound on the cost of travelling (on any possible path) between v and any point on a boundary edge not incident to v. For any boundary edge $e = \overline{v_1 v_2}$, u_e divides edge e into two segments $\overline{v_1 u_e}$ and $\overline{u_e v_2}$, as shown in Fig. 2. The first Steiner point $u_{i,1}$ is placed on segment $\overline{v_i u_e}$ with distance $b_e R_{v_i} \gamma$ to vertex v_i. The subsequent Steiner point $u_{i,j}$ is placed between $u_{i,j-1}$ and u_e with a distance $\gamma b_e \tau_{\min}(u_{i,j-1})$ to $u_{i,j-1}$. We continue adding Steiner points until no more Steiner point can be added on $\overline{v_i u_e}$. That is, if $u_{i,j}$ is the last Steiner point added, no more Steiner point is inserted on segment $\overline{v_i u_e}$ if $|\overline{v_i u_{i,j}}| + \gamma b_e \tau_{min}(u_{i,j}) \geq \overline{v_i u_e}$. Finally, we add u_e as a Steiner point on e.

Theorem 1 (Adapted from theorem 5 in [14]). *For the discretization scheme in [14], the total number of Steiner points added to the triangular subdivision is* $O(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma})$.

2.2 Constructing the Graph

Given a source point s, a destination point t, a positive real value ε, and a triangular subdivision, we will show how to build the graph G'. Let r be the region where the source point s lies. If s is neither a boundary point nor a subdivision vertex, then triangulate r into three triangles with apex at s. The same can be done if point t is neither a Steiner point nor a subdivision vertex. Then place Steiner points in geometric progression along the new edges (as described in Section 2.1). Once the Steiner points are placed, we can construct a weighted directed graph G'. Throughout this paper we will use $G = (V, E)$ to denote the graph constructed by Reif and Sun [14]. The main problem in [14] is the quadratic complexity of G with respect to the number of Steiner points. Our main target is to maintain a linear number of edges in each region. Since it suffices to approximate the quickest paths, we will show how the WSPD [4] can help us to achieve the goal.

Definition 1 ([4]). *Let $s > 0$ be a real number, and let A and B be two finite sets of points in $\mathbb{R}^d$. We say that A and B are* well-separated *with respect to s, if there are two disjoint d-dimensional balls C_A and C_B, having the same radius, such that (i) C_A contains the bounding box $R(A)$ of A, (i) C_B contains the bounding box $R(B)$ of B, and (ii) the minimum distance between C_A and C_B is at least s times the radius of C_A.*

The parameter s will be referred to as the *separation constant*. The next lemma follows easily from Definition 1.

Lemma 2 ([4]). *Let A and B be two finite sets of points that are well-separated w.r.t. s, let x and p be points of A, and let y and q be points of B. Then (i) $|xy| \leq (1 + 2/s) \cdot |xq|$, (i) $|xy| \leq (1 + 4/s) \cdot |pq|$, and (ii) $|px| \leq (2/s) \cdot |pq|$.*

Definition 2 ([4]). *Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A* well-separated pair decomposition *(WSPD) for S with respect to s is a sequence of pairs of non-empty subsets of S, $(A_1, B_1), \ldots, (A_m, B_m)$, such that*

1. *$A_i \cap B_i = \emptyset$, for all $i = 1, \ldots, m$,*
2. *for any two distinct points p and q of S, there is exactly one pair (A_i, B_i) in the sequence, such that (i) $p \in A_i$ and $q \in B_i$, or (ii) $q \in A_i$ and $p \in B_i$,*
3. *A_i and B_i are well-separated w.r.t. s, for $1 \leq i \leq m$.*

The integer m is called the size *of the WSPD.*

Callahan and Kosaraju showed that a WSPD of size $m = \mathcal{O}(s^d n)$ can be computed in $\mathcal{O}(s^d n + n \log n)$ time.

Observation 1. *Let $r = \triangle ABC$ be a triangular region such that $\overline{AB}$ is the diagonal and $\overline{AC} > \overline{BC}$, then $\overline{AC} > \frac{1}{2}\overline{AB}$.*

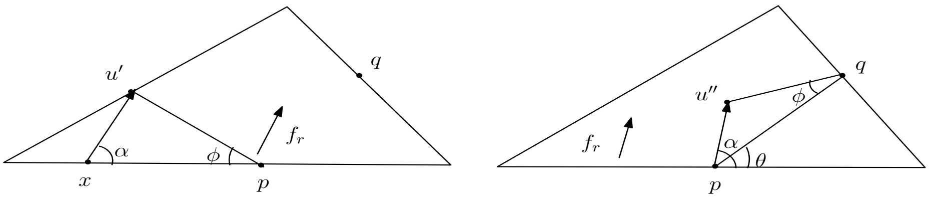


Fig. 3. Illustrating the proof of Lemma 3

Lemma 3. *Let A and B be two finite sets of Steiner points that are well-separated w.r.t. s and let x and p be points of A, and let q be a point of B (see figure 3). Then $\tau(x,p) \leq \frac{4}{s} \cdot \tau(p,q)$ and $\tau(p,x) \leq \frac{4}{s} \cdot \tau(p,q)$.*

The proof is straight-forward and is omitted in this abstract.

Let V_j be the set of region vertices and Steiner points of region r_j, $1 \leq j \leq m$. For each region r_j, compute a WSPD $\{(A_i, B_i)\}_{i=1}^{k}$ of V_j with respect to a separation constant $s = \frac{32}{\gamma}$. Next, construct the graph $G' = (V, E')$, where V is the set of Steiner points and vertices of the triangular subdivision (see Section 2.2) and E' is constructed as follows. For each well-separated pair $\{(A_i, B_i)\}$ in r_j, pick two arbitrary points $a \in A_i$ and $b \in B_i$ as representative points. Add the directed edges (a_j, b_j) and (b_j, a_j) to E with weights $\tau(a_j, b_j)$ and $\tau(b_j, a_j)$ respectively.

Lemma 4. *The graph G' has $O(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma})$ vertices and $O(s^d \cdot \frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma})$ edges and can be built in $O(s^d \cdot (\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma}) \log(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma}))$ time.*

Proof. The graph G' has $O(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma})$ vertices according to the discretization scheme used in [14]. The number of edges is linear with respect to the number of vertices, which follows from using the WSPD that requires $O(s^d \cdot (\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma}) \log(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma}))$ time to be computed. $\square$

By simply running Dijkstra's algorithm [6], implemented using Fibonacci heaps, on G' gives the following theorem.

Theorem 2. *A quickest path between s and t in G' can be computed in $O(s^d \cdot (\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma}) \log(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma}))$ time using $O(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma})$ space.*

Our algorithm improves Reif and Sun algorithm [14] in terms of time complexity by roughly a factor of $\frac{c_{skew}}{\gamma}$. The reason for the improvement is the use of the WSPD as it reduces the complexity of the number of edges from quadratic to linear within a region.

2.3 Bounding the Error of the Approximation

Let G and G' be the two graphs as described above. Recall that V is the set of Steiner points and vertices of the triangular subdivision and E is constructed by connecting every pair of Steiner points and vertices in the same region of the subdivision (including s and t). Also recall that E' is constructed by connecting every pair of representative points in the same region. Here we analyze how well a shortest path in G' approximates a shortest path in G from a given source point s to a given destination point t.

Let $\mathcal{P}$ be an optimal path between s and t in the continuous space with cost of $D(\mathcal{P})$ and let $\mathcal{P}_1$ be a shortest path in G between the two vertices corresponding to s and t. Denote the cost of $\mathcal{P}_1$ by $D(\mathcal{P}_1)$ and let $S = \{b_1, b_2, ..., b_{m-1}, b_m\}$ be the points visited by $\mathcal{P}_1$ in order of occurrence. Let $\delta_G(p, q)$ be the cost of a quickest path in G between two points p and q and let $\delta_{G'}(p, q)$ be the cost of a quickest path between p and q in G'. The following theorem is adapted from Theorem 6 in [14].

Theorem 3 (Adapted from Theorem 5 in[14]). *For any piecewise linear path $\mathcal{P}$ from a source point $s \in V$ to a destination point $t \in V$, there exist a discrete path $\mathcal{P}_1$ from s to t in G such that $D(\mathcal{P}_1) \leq (1 + \gamma) \cdot D(\mathcal{P})$.*

Consider each segment $\overline{b_i b_{i+1}}$, $1 \leq i < m$, along $\mathcal{P}_1$. Recall that b_i, b_{i+1} must be points in V but they may not be connected by an edge in G'. However, for every pair b_i, b_{i+1} there exists a well-separated pair (A, B) such that $a_i, b_i \in A$, $a_{i+1}, b_{i+1} \in B$ and, a_i and a_{i+1}) are connected by a directed edge. In the next lemma we will show that for every segment along $\mathcal{P}_1$ there exists a path in G' that is almost as quick as the segment.

Lemma 5. *Let p and q be any pair of points in V that lie in the same region r, it holds that $\delta_{G'}(p,q) \leq (1+\frac{32}{s}) \cdot \delta_G(p,q)$, where $s > 16$ is the separation constant of the WSPD.*

Proof. The proof is done by induction on the Euclidean distance between p and q. We know that there exists a well-separated pair (A,B) such that $p \in A$ and $q \in B$, and that there exists an edge from p' to q' in G' and in region r with $p' \in A$ and $q' \in B$. If the separation constant for the WSPD is s then we have that $|\overline{pp'}| \leq \frac{2}{s} \cdot |\overline{p'q'}|$ and $|\overline{q'q}| \leq \frac{2}{s} \cdot |\overline{p'q'}|$, which follows from Lemma 2.

Base case: Assume that (p,q) is the closest pair of V. In this case there exists a well-separated pair $\{(A,B)\}$ such that $A = \{p\}$ and $B = \{q\}$, otherwise (p,q) could not be the closest pair. Hence the claim holds since $p = p'$ and $q = q'$ there must also be an edge in G' from p to q.

Induction hypothesis: Assume that the lemma holds for all pairs in V closer than $|\overline{pq}|$ to each other.

Induction step: According to Lemma 2, $|pp'| < |p'q'|$ and $|q'q| < |p'q'|$. According to the induction hypothesis there is a path $\delta_{G'}(p,p')$ with cost at most $(1+32/s) \cdot \delta_G(p,p')$ and a path $\delta_{G'}(q',q)$ with cost at most $(1+32/s) \cdot \delta_G(q',q)$. Also, recall that the cost of the edge (p',q') in G' is $\delta_G(p',q') = \tau(p',q')$. We get:

$$\begin{aligned}
\delta_{G'}(p,q) &\leq \delta_{G'}(p,p') + \delta_{G'}(p',q') + \delta_{G'}(q',q) \\
&< (1+32/s) \cdot \delta_G(p,p') + (\delta_G(p,p') + \delta_G(p',q') + \delta_G(q',q)) \\
&\quad +(1+32/s) \cdot \delta_G(q',q) \\
&\leq (2+32/s) \cdot \tau(p,p') + \tau(p,q) + (2+32/s) \cdot \tau(q',q) \\
&\leq 4/s \cdot (2+32/s) \cdot \tau(p,q) + \tau(p,q) + 4/s \cdot (2+32/s) \cdot \tau(p,q) \\
&\leq (1+16/s+256/s^2) \cdot \delta_G(p,q) \\
&< (1+32/s) \cdot \delta_G(p,q)
\end{aligned}$$

□

From Lemma 5, we can now to establish the following theorem:

Theorem 4. *For any piecewise linear path $\mathcal{P}$ from a source point $s \in V$ to a destination point $t \in V$, we have $\delta'_G(s,t) \leq (1+\varepsilon) \cdot D(\mathcal{P})$.*

Proof.

$$\begin{aligned}
\delta_{G'}(s,t) &\leq \sum_{j=1}^{m} (1+32/s) \cdot \delta_G(b_j, b_{j+1}) \\
&= (1+32/s) \cdot \delta_G(s,t) \\
&\leq (1+32/s)(1+\gamma)D(\mathcal{P})
\end{aligned}$$

By setting $s = \varepsilon/32$ and $\gamma = \varepsilon/4$ the theorem follows since $\varepsilon < 1$. □

3 Shortest Path Queries

In this section we turn our attention to the query version. That is, preprocess the subdivision $\mathcal{T}$ such that given any two points s and t in $\mathbb{R}^2$ find a quickest path between s and t in $\mathcal{T}$. We present a data structure that, given two query points s and t, and a positive real value ε, returns a path between s and t whose weight is at most $(1+\varepsilon)$ times the weight of an optimal path between s and t.

We will start with the simpler case, when t is already known in advance and we are only given the source point s and $\varepsilon > 0$ as query. Then, in Section 3.3 we will show how to generalize it to the case when both s and t are given as a query.

3.1 The Preprocessing and the Query

In this subsection we will present the data structure, describe the preprocessing and show how a query is processed. Throughout this section, we assume that s lies entirely inside a region. If s lies on the boundary then we can just perturb it slightly.

Preprocessing. In the preprocessing step we need to build three data structures, denoted M, N and P.

$\mathbf{M}[\cdot]$: In Section 2.2 we showed how to build a graph G' given a triangular subdivision and two points. Build the same graph, again denoted $G'(V, E')$, but without including the source point s. Then compute the quickest path in G', using Dijkstra's shortest path algorithm, from every vertex in V to t. Note that this can be done by a single call to Dijkstra's algorithm from t to all other vertices in V provided that the directions of all edges have been reversed. The costs are stored in a vector M, such that for a vertex $v \in V$ the entry $M[v]$ stores the cost of the quickest path in G' from v to t.

According to Lemma 4 the complexity of G' is linear with respect to the number of vertices, thus it takes $O(s^d(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma}) \log(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma}))$ time and requires $O(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma})$ space to build $M[\cdot]$ [6]. For reasons that will become clear below we set $s = \max\{16, \varepsilon/64\}$ and $\gamma = \varepsilon/4$.

$\mathbf{N_r}[\cdot,\cdot]$: The second structure is an angle restricted nearest neighbor querying structure for each face r in the subdivision. Let $\kappa \geq 9$ (follows from the construction of θ-graphs) and let $\theta = 2\pi/\kappa$. If we rotate the positive x-axis by angles $i\theta$, $0 \leq i < \kappa$, then we get κ rays, denoted $\langle r_1, \ldots, r_\kappa \rangle$. Each pair of successive rays r_i and r_{i+1}, $1 \leq i < \kappa$, defines a cone X_i whose apex is at the origin. The cone obtained by translating X_i such that its apex is at a point q is denoted $X_i(q)$.

Given a set $\mathcal{S}$ of n points in the plane we build a data structure $N[q, i]$ that given a query point $q \in \mathbb{R}^2$ and an integer i, $0 \leq i < \kappa$, returns a point s of $\mathcal{S}$ within the cone $X_i(q)$ whose orthogonal projection s_p onto the bisector of $X_i(q)$ is the smallest. It has been shown (see for example Section 4.1.2 in [12] or Lemma 2 in [3]) that such a structure can be preprocessed in $O(\kappa n \log n)$ time into a data structure of size $O(\kappa n)$ such that queries can be answered in $O(\log n)$ time. For our purposes we will set $\kappa = 50/\varepsilon$.

$\mathbf{P}[\cdot]$: Finally, the triangular subdivision is preprocessed for efficient point location queries as described in Chapter 6.1 in [2], which can be performed in time $O((\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma}) \log(\frac{c_{skew} \cdot n}{\gamma} \log \frac{c_{skew}}{\gamma}))$. That is, given a query point q the data structure P returns the triangle in the subdivision that contains q.

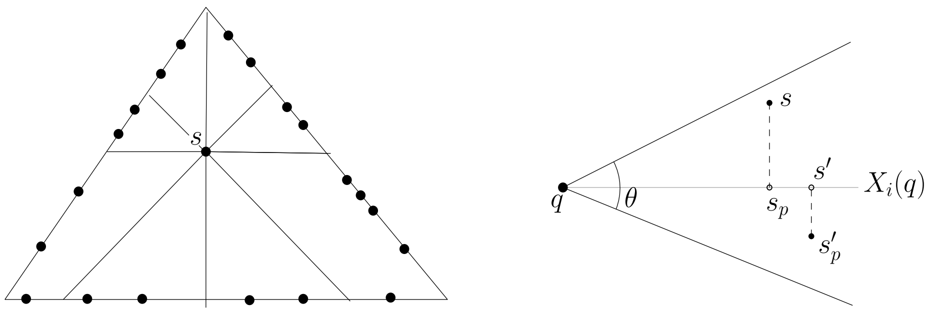


Fig. 4. (a) Partitioning the plane into κ cones. (b) Selecting the point whose orthogonal projection onto the bisector of $X_i(q)$.

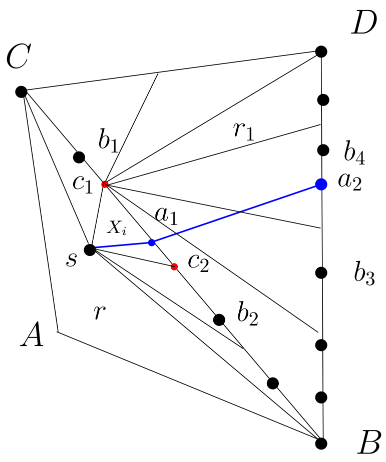


Fig. 5. Handling the case of the empty cone X_i

Theorem 5. *Given a positive constant ε, the preprocessing requires $O(s^d \cdot (\frac{c_{skew} \cdot n}{\varepsilon} \log \frac{c_{skew}}{\varepsilon}) \log(\frac{c_{skew} \cdot n}{\varepsilon} \log \frac{c_{skew}}{\varepsilon}))$ time and $O(\frac{c_{skew} \cdot n}{\varepsilon} \log \frac{c_{skew}}{\varepsilon})$ space.*

Query. As a query we are given a point s in the plane. Perform a point location query $\mathcal{P}[q]$ to determine which region r contains s. Next, for each i, $1 \leq i < \kappa$ perform an angle constrained nearest neighbour query $N_r[s, i]$, as shown in Fig. 4. Consider a cone $X_i(s)$. We have two cases: (1) $X_i(s)$ contains a point of V, or (2) $X_i(s)$ is empty.

Recall that the data structure M stores all the quickest paths to t from any point in V. Thus, our task is to find a "good" point v in V such that $\tau(s, v) + M[v, t]$ approximates $D(\mathcal{P})$.

Case 1: For each non empty cone $X_i(s)$, $1 \le i < \kappa$, let $v_i := N_r[q,i]$, see Fig. 4. Next, compute the cost of the quickest path $\mathcal{P}'_i$ in G' between s and t via v_i, that is, $\tau(s, v_i) + M[v_i]$.

Case 2: For each empty cone $X_i(s)$, $1 \le i < \kappa$, let $\overline{CB}$ be a boundary edge of region r that intersects with $X_i(s)$, as shown in Fig. 5. If the cone intersects several boundary edges just pick one of them. Let r_1 be the region that shares a boundary edge $\overline{CB}$ with region r. Let c_1 and c_2 be the points of intersection between the two boundaries of $X_i(s)$ and $\overline{CB}$, such that c_1 is closer, or as close, to s as c_2, as shown in Fig. 5. Note that c_1 and c_2 are not points in V.

For each cone $X_i(c_1)$, $1 \le j < \kappa$, let $v_j := N_r[q,j]$. Compute the cost of the quickest path in G' between s and t via v_j, that is, $\tau(s,c_1)+\tau(c_1,v_j)+M[v_j]$. The quickest among these paths is denoted $\mathcal{P}'_{i,j}$, that is $\mathcal{P}'_i$ is the path that minimizes $\tau(s,c_1) + \tau(c_1,v_j) + M[v_j]$ among all v_j.

To conclude the query report the path, denoted $\mathcal{P}'$, between s and t that has the minimum travelling time among all computed paths, that is, $\mathcal{P}'$ is the path quickest among all paths $\mathcal{P}'_i$, $1 \le i \le \kappa$.

The approximation bound and the query time will be proven in the next section.

3.2 Approximation Bound

We establish the following theorem that provides a bound on the error of using a discrete path to approximate an optimal path. As mentioned earlier, we have two cases for the query either (1) the cone where the first edge along $\mathcal{P}$ lies in contains a Steiner point or subdivision vertex, or (2) the cone where the first edge along $\mathcal{P}$ lies in is empty. In the following we show that $D(\mathcal{P}')$ is no more than $(1+\varepsilon) \cdot D(\mathcal{P})$ for the two cases by using several lemmas.

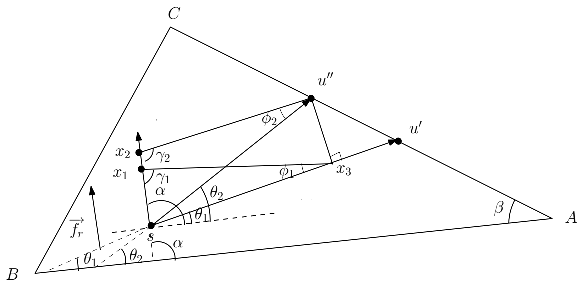


Fig. 6. Illustrating the notations in the proof of Lemma 6

Lemma 6. *Let $X_i(s)$ be the cone where the first segment along $\mathcal{P}$ lies in. If $X_i(s)$ contains a vertex in V then the algorithm returns a path $\mathcal{P}'$ such that $D(\mathcal{P}') \le (1+\varepsilon) \cdot D(\mathcal{P})$.*

Proof. The proof is omitted in this abstract. □

It remains to consider Case 2, if the cone where the first segment along $\mathcal{P}$ lies in is empty. Assume without loss of generality that b_1 and b_2 are the nearest Steiner points to c_1 and c_2 respectively on the boundary edge $\overline{CB}$ as shown in Fig. 5. Let $a_1, \ldots, a_m$ be the points where $\mathcal{P}$ bends. The following lemma is adapted from Lemma 14 in [14]. The construction of the set of Steiner points is described in Section 2.1.

Lemma 7 (Adapted from Lemma 15 in [14]). *Let $r = \triangle BCD$ be a region containing a path segment $\overline{a_i a_{i+1}}$ of an optimal path $\mathcal{P}$. Assume that a_i lies between two Steiner points b_1 and b_2 on the boundary edge $\overline{CB}$ and a_{i+1} lies between two Steiner points b_3 and b_4 on the boundary edge $\overline{DB}$, as shown in Fig. 5. Recall that u_e is the point on the boundary edge e with maximum $\tau_{\min}(u)$. Given a positive constant ϕ and a destination point t in V then, if b_1 lies between C and u_e then $\tau(b_1, b_3) < (1 + \phi) \cdot \tau(a_1, a_2)$ and $\delta_G(b_3, t) < (1 + \phi) \cdot \tau(a_2, t)$ otherwise $\tau(b_2, b_4) < (1 + \phi) \cdot \tau(a_1, a_2)$ and $\delta_G(b_4, t) < (1 + \phi) \cdot \tau(a_2, t)$.*

Lemma 8. *Let $X_i(s)$ be a cone that contains the first segment along $\mathcal{P}$. If $X_i(s)$ is empty then there exists a path $\mathcal{P}'$ in G such that $D(\mathcal{P}') \le (1 + \varepsilon) \cdot D(\mathcal{P})$.*

Proof. According to the construction of $\mathcal{P}'$ we have $D(\mathcal{P}') = \tau(s, c_1) + \tau(c_1, v_j) + \delta_{G'}(v_j, t)$ while the optimal path can be described as $D(\mathcal{P}) = \tau(s, a_1) + \tau(a_1, a_2) + \tau(a_2, t)$.

The proof is divided into parts. In the first part we prove that $\tau(s, c_1) \le (1 + \varepsilon) \cdot \tau(s, a_1)$ and in the second part we prove that $\tau(c_1, v_j) + \delta_{G'}(v_j, t) \le (1 + \varepsilon) \cdot (\tau(a_1, a_2) + \tau(a_2, t))$. The first part can be proved using the same arguments used in Lemma 6 so we will omit it here. The second part can be proved as follows. Without loss of generality we assume that b_1 lies between C and u_e, and using Lemma 7 we have:

$$\tau(b_1, b_3) < (1 + \phi) \cdot \tau(a_1, a_2) \quad \text{and} \quad \delta_G(b_3, t) < (1 + \phi) \cdot \tau(a_2, t). \tag{1}$$

Recall that to compute the second segment, denoted (c_1, v''), along $\mathcal{P}'$ we partition the region r_1 into κ cones with apex at c_1. Then using the same arguments as in Lemma 6 together with (1), we can prove the following:

$$\tau(c_1, v'') < (1 + \phi) \cdot \tau(a_1, a_2). \tag{2}$$

From Lemma 7 and Theorem 4 we get: $\delta_{G'}(b_3, t) < (1 + \varepsilon) \cdot \tau(a_2, t)$. Using exactly the same argument we can prove: $\delta_{G'}(b_4, t) < (1 + \varepsilon) \cdot \tau(a_2, t)$. which also implies that

$$\delta_{G'}(v'', t) < (1 + \varepsilon) \cdot \tau(a_2, t). \tag{3}$$

By setting $\phi = \varepsilon$ and then using (2) and (3) we get:

$$\tau(c_1, v'') + \delta_{G'}(v'', t) < (1 + \varepsilon) \cdot (\tau(a_1, a_2) + \tau(a_2, t)) \tag{4}$$

Finally putting together the first and second part of the proof we obtain the desired result $D(\mathcal{P}') \le (1 + \varepsilon) \cdot D(\mathcal{P})$. □

Theorem 6. *Given a planar triangular subdivision $\mathcal{T}$ with a translational flow of complexity n, a point $t \in \mathbb{R}^2$ and a positive constant ε, one can preprocess $\mathcal{T}$ in $O((\frac{c_{skew} \cdot n}{\varepsilon}) \log \frac{c_{skew}}{\varepsilon}) \log(\frac{c_{skew} \cdot n}{\varepsilon} \log \frac{c_{skew}}{\varepsilon}))$ time using $O((\frac{c_{skew} \cdot n}{\varepsilon}) \log \frac{c_{skew}}{\varepsilon})$ space such that given a query point $s \in \mathbb{R}^2$ a $(1+\varepsilon)$-approximate quickest path between s and t can be calculated in $O(1/\varepsilon^2 \cdot \log(\frac{c_{skew} \cdot n}{\varepsilon} \log \frac{c_{skew}}{\varepsilon}))$ time.*

3.3 General Case

In this section we turn our attention to the query version when we are given two query points s and t in $\mathbb{R}^2$ and our goal is to find the quickest path between s and t. The idea is the same as in the previous section. That is we perform the exact same preprocessing steps as in the previous section (omitting the destination point t), but with the exception that M contains all-pair quickest costs. Using Johnson's algorithm [7] the all-pairs shortest paths can be computed in $O(s^d(\frac{c_{skew} \cdot n}{\varepsilon})^2 \log \frac{c_{skew}}{\varepsilon}) \log(\frac{c_{skew} \cdot n}{\varepsilon} \log \frac{c_{skew}}{\varepsilon}))$ using $O((\frac{c_{skew} \cdot n}{\varepsilon})^2 \log \frac{c_{skew}}{\varepsilon})$ space.

The query is performed as in the previous section, however, for both s and t one needs to find a "good" next point along the path. That is, instead of only searching for a point in each cone $X_i(s)$, we also need to search for a "good" point in each $X_i(t)$ and the try all combinations of the two. By putting together the results, we obtain the following theorem:

Theorem 7. *Given a planar triangular subdivision $\mathcal{T}$ of complexity n with a translational flow and a positive constant ε, one can preprocess $\mathcal{T}$ in time $O((\frac{c_{skew} \cdot n}{\varepsilon})^2 \log \frac{c_{skew}}{\varepsilon}) \log(\frac{c_{skew} \cdot n}{\varepsilon} \log \frac{c_{skew}}{\varepsilon}))$ using $O((\frac{c_{skew} \cdot n}{\varepsilon})^2 \log \frac{c_{skew}}{\varepsilon})$ space such that given two query points s and t a $(1+\varepsilon)$-approximate quickest path between s and t can be calculated in $O(1/\varepsilon^4 \cdot \log(\frac{c_{skew} \cdot n}{\varepsilon} \log \frac{c_{skew}}{\varepsilon}))$ time.*

4 Concluding Remarks

We considered the problem of computing a quickest path in a subdivision with a translational flow. In the basic case we presented an algorithm that slightly improves the running time of the algorithm by Reif and Sun [14]. Our main contribution is an effective data structure for the query version of the problem.

There are many open problems remaining. For example, can one develop a more efficient data structure that has a smaller dependency on the triangulation? We believe our construction can be generalized to higher dimensions, but at what cost?

References

1. Aleksandrov, L., Lanthier, M., Maheshwari, A., Sack, J.-R.: An epsilon-Approximation for Weighted Shortest Paths on Polyhedral Surfaces. In: Proceedings of the 6th Scandinavian Workshop on Algorithm Theory (1998)
2. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, Heidelberg (2008)

3. Bose, P., Gudmundsson, J., Morin, P.: Ordered theta graphs. Computational geometry – Theory & Applications 28(1), 11–18 (2004)
4. Callahan, P.B., Kosaraju, S.R.: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. Journal of the ACM 42, 67–90 (1995)
5. Cheng, S.-W., Na, H.-S., Vigneron, A., Wang, Y.: Approximate Shortest Paths in Anisotropic Regions. SIAM Journal on Computing 38(3), 802–824 (2008)
6. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
7. Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. Journal of the ACM 24(1), 1–13 (1977)
8. Lanthier, M., Maheshwari, A., Sack, J.-R.: Approximating Weighted Shortest Paths on Polyhedral Surfaces. In: Proceedings of the 13th Symposium on Computational Geometry, pp. 274–283 (1997)
9. Mata, C., Mitchell, J.: A New Algorithm for Computing Shortest Paths in Weighted Planar Subdivisions. In: Proceedings of the 13th Symposium on Computational Geometry, pp. 264–273 (1997)
10. Mitchell, J.: Geometric shortest paths and network optimization. Handbook of Computational Geometry, 633–701 (2000)
11. Mitchell, J., Papadimitriou, C.: The weighted region problem: Finding shortest paths through a weighted planar subdivision. Journal of the ACM 38(1), 18–73 (1991)
12. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, Cambridge (2007)
13. Papadakis, N., Perakis, A.: Deterministic Minimal Time Vessel Routing. Journal of Operations Research 38(3), 426–438 (1990)
14. Reif, J., Sun, Z.: Movement Planning in the Presence of Flows. Algorithmica 39(2), 127–153 (2004)
15. Rowe, N.: Obtaining Optimal Mobile-Robot Paths with Nonsmooth Anisotropic Cost Functions Using Qualitative-State Reasoning. International Journal of Robotics Research 16(3), 375–399 (1997)

Mechanisms for Obnoxious Facility Game on a Path

Yukun Cheng[1,2], Wei Yu[1], and Guochuan Zhang[1]

[1] College of Computer Science and Technology, Zhejiang University, Hangzhou, 310027, China

[2] School of Mathematics and Statistics, Zhejiang University of Finance and Economics, Hangzhou 310018, China

ykcheng@amss.ac.cn, {zyuwei2006831,zgc}@zju.edu.cn

Abstract. We consider a new facility game, namely, an obnoxious facility game where the facility is undesirable and all agents try to be as far away from the facility as possible. The social cost is the total distance between the agents and the facility. However, an obnoxious facility is placed based on the reported locations of the selfish agents. We are interested in a mechanism to decide the facility location so that the social cost is maximized. In this paper, we give a first attempt for this game on a path. Our main results include a 3-approximation group strategy-proof deterministic mechanism, which is best possible if the facility can only take one of the endpoints on the path, and two group strategy-proof randomized mechanisms with approximation ratio of $\frac{5}{3}$ and $\frac{3}{2}$, respectively.

Keywords: Algorithmic mechanism design, facility location, social choice.

1 Introduction

In this paper, we discuss a new facility game, called *obnoxious facility game*. Consider the following scenario. The local government plans to build a garbage dump in an area. All residents report their home addresses so that the government can decide the most appropriate location. The cost of each resident is her distance form the garbage dump, and the obnoxious social cost is defined to be the total cost of all residents. Since the garbage dump is not popular, every resident wants to be as far to the garbage dump as possible, and the government aims to maximize the obnoxious social cost. Different from the traditional location problems, in our setting the location of the residents is private information, where the residents may report wrong locations to maximize their distance to the facility. The core of this game is to design a mechanism (algorithm) which decides the facility location based on the residents' reports. We are interested in mechanisms which are group strategy-proof, in the sense that no group of residents can misreport their locations such that each member can strictly benefit.

The previous work was concerned about the classical facility game, in which each agent (resident) wants to stay as close to a facility as possible. Procaccia and Tennenholtz [6] considered the facility game when all agents are located on a line.

For the 1-facility game, it is trivial that there exists an optimal group strategy-proof mechanism. For the 2-facility game on a line, they gave an upper bound of $n-2$ and a lower bound of 1.5 for deterministic strategy-proof mechanisms. Later, Lu et al. [4] obtained an upper bound of $\frac{n}{2}$ and a lower bound of 1.045 for randomized strategy-proof mechanisms. Recently, Lu et al. [3] improved the lower bound for deterministic strategy-proof mechanisms to $\frac{n-1}{2}$ and designed a 4-approximation randomized mechanism in general metric spaces. Alon et al. [1] studied the facility game with one facility in more general networks rather than a line. They gave an almost full picture of the feasible strategy-proof mechanisms.

Our work is the first try to study the strategy-proof approximation mechanisms for obnoxious facility games. In our basic setting, the agents are located on a path with left and right endpoints a and b, and the mechanism must select the location of an obnoxious facility. We first give a 3-approximation group strategy-proof deterministic mechanism. We then show that if the facility is restricted to the endpoints, then no mechanism can do better. But if the facility can be located on the interior point of the interval, the approximation ratio of any strategy-proof deterministic mechanism is at least 2. Finally, we design two group strategy-proof randomized mechanisms with approximation ratios of $\frac{5}{3}$ and $\frac{3}{2}$, respectively, which break the deterministic lower bound of 2.

2 Preliminaries

We use the same definitions and notions as in [3]. Let $N = \{1, 2, \cdots, n\}$ be the set of agents. All of the agents are located on a path P. For the sake of simplicity, without loss of generality, assume that the left endpoint of the path is zero and the right endpoint of the path is two. We can regard the path as an interval $I = [0, 2]$. The distance between any two points $x, y \in I$ is $d(x, y) = |x - y|$. Thus for all $x \in I$, $d(x, x) = 0$. The location reported by agent i is $x_i \in I$. We denote $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ a location profile.

In the obnoxious facility game, a deterministic mechanism outputs a facility location based on a given location profile $\mathbf{x}$ and thus is a function $f : I^n \to I$. Assuming the facility location to be $y = f(\mathbf{x})$, the cost of agent i is her distance to the facility, i.e.,

$$cost(f(\mathbf{x}), x_i) = |y - x_i|.$$

A randomized mechanism is a function $f : I^n \to \Delta(I)$, where $\Delta(I)$ is the set of distributions over I. The cost of agent i is now her expected cost over such distribution

$$cost(f(\mathbf{x}), x_i) = E_{y \sim f}\big[|y - x_i|\big].$$

Let $\mathbf{x}_{-i} = (x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$ be the location profile without agent i. For an agent set $S \subseteq N$, we denote $\mathbf{x}_S$ and $\mathbf{x}_{-S}$ to be the location profiles of agents in and outside S, respectively. Thus we have three equivalent notations: $\mathbf{x} = \langle x_i, \mathbf{x}_{-i} \rangle = \langle \mathbf{x}_S, \mathbf{x}_{-S} \rangle$. For simplicity, we write $f(x_i, \mathbf{x}_{-i}) = f(\langle x_i, \mathbf{x}_{-i} \rangle)$ and

$f(\mathbf{x}_S, \mathbf{x}_{-S}) = f(\langle \mathbf{x}_S, \mathbf{x}_{-S} \rangle)$. The *obnoxious social cost* of a mechanism f on a location profile $\mathbf{x}$ is defined as the total cost of n agents

$$SC(f, \mathbf{x}) = \sum_{i=1}^{n} cost(f(\mathbf{x}), x_i).$$

In the randomized case, this obnoxious social cost is an expected value.

For the obnoxious facility game, we are interested in strategy-proof mechanisms that also do well with respect to maximizing the obnoxious social cost. For a location profile $\mathbf{x}$, let $\mathrm{OPT}(\mathbf{x})$ be the optimal social cost. We say a mechanism f has an approximation ratio γ, if for all profile $\mathbf{x} \in I^n$,

$$\mathrm{OPT}(\mathbf{x}) \leq \gamma SC(f, \mathbf{x}).$$

In the following we formally define the strategy-proofness and the group strategy-proofness.

Definition 1. *A mechanism is strategy-proof if no agent can benefit from misreporting her location. Formally, given agent i, profile $\mathbf{x} = \langle x_i, \mathbf{x}_{-i} \rangle \in I^n$, and a misreported location $x_i' \in I$, it holds that*

$$cost(f(x_i, \mathbf{x}_{-i}), x_i) \geq cost(f(x_i', \mathbf{x}_{-i}), x_i).$$

Definition 2. *A mechanism is group strategy-proof if for any group of agents, at least one of them cannot benefit if they misreport simultaneously. Formally, given a non-empty set $S \subseteq N$, profile $\mathbf{x} = \langle \mathbf{x}_S, \mathbf{x}_{-S} \rangle \in I^n$, and the misreported location $\mathbf{x}_S' \in I^{|S|}$, there exists $i \in S$, satisfying*

$$cost(f(\mathbf{x}_S, \mathbf{x}_{-S}), x_i) \geq cost(f(\mathbf{x}_S', \mathbf{x}_{-S}), x_i).$$

3 Deterministic Mechanisms

Taking the algorithmic perspective, our work is related to the literature on approximation algorithms for 1-maxian problem [10,8,9,2]. Thus it is well-known that, when n agents are located on an interval $[0, 2]$, one of two endpoints must be an optimal facility location. Hence, we have that $\mathrm{OPT}(\mathbf{x}) = \max\{\sum_{i=1}^{n} x_i, \sum_{i=1}^{n}(2 - x_i)\}$ for a location profile $\mathbf{x}$. According to the optimal cost, a deterministic mechanism can be devised directly. That is, if $\sum_{i=1}^{n} x_i \geq \sum_{i=1}^{n}(2 - x_i)$ then return the facility location $y = 0$; otherwise $y = 2$. Unfortunately, this mechanism is not strategy-proof. Indeed, if $N = \{1, 2\}$, $x_1 = \frac{2}{3}$ and $x_2 = \frac{6}{5}$. Clearly, the mechanism returns the facility location $y = 2$ since $\frac{2}{3} + \frac{6}{5} < (2 - \frac{2}{3}) + (2 - \frac{6}{5})$. But agent 2 can move the facility location to the left endpoint by reporting $x_2' = 1$. In the following we present a group strategy-proof deterministic mechanism.

Mechanism 1. Given location profile $\mathbf{x}$ on interval $[0, 2]$. Let n_1 be the number of agents located on $[0, 1]$ and n_2 be the number of agents on $(1, 2]$. Obviously, $n_1 + n_2 = n$. If $n_1 \leq n_2$, then return the left endpoint 0; otherwise return the right endpoint 2.

Theorem 1. *Mechanism 1 is a group strategy-proof* 3*-approximation mechanism for the obnoxious facility game when the agents are located on a path.*

Proof. Given a location profile $\mathbf{x}$ on interval [0,2]. First we shall prove the claim about the approximation ratio. We only discuss the case that $n_1 \leq n_2$. The proof for the other case is similar. If $n_1 \leq n_2$, Mechanism 1 returns the facility location $y = 0$. Thus the obnoxious social cost is

$$SC(f,\mathbf{x}) = \sum_{i=1}^{n} cost(f(\mathbf{x}), x_i) = \sum_{i=1}^{n} x_i \geq \sum_{x_i \in (1,2]} x_i \geq n_2 \geq \frac{n}{2}.$$

If $\text{OPT}(\mathbf{x}) = \sum_{i=1}^{n}(2 - x_i)$, then we have

$$\text{OPT}(\mathbf{x}) = \sum_{i=1}^{n}(2 - x_i) \leq 2n_1 + n_2 = n + n_1 \leq \frac{3}{2}n \leq 3SC(f,\mathbf{x}).$$

Let us turn to show the group strategy-proofness of Mechanism 1. Let $S \subseteq N$ be a coalition. We must demonstrate that the agents in S cannot all gain by deviating. Without loss of generality, we assume that $n_1 \leq n_2$. Hence, Mechanism 1 outputs the facility location $y = 0$. Any agent in S misreports her location x_i to x'_i. Let n'_1 and n'_2 be the number of agents on $[0,1]$ and $(1,2]$ after misreporting, respectively. The new profile is $\mathbf{x}'$ and the new facility location is denoted by y'. It is easy to know that if $n'_1 \leq n'_2$, then $y' = y = 0$ and $cost(f(\mathbf{x}), x_i) = cost(f(\mathbf{x}'), x_i)$ for any agent $i \in N$. If $n'_1 > n'_2$, then $y' = 2$. And at least one agent on $(1,2]$ misreports her location to $x'_i \in [0,1]$. Thus we have $cost(f(\mathbf{x}'), x_i) = 2 - x_i < 1 < cost(f(\mathbf{x}), x_i)$. □

Since one of endpoints must be an optimal solution, we only consider the left and right endpoint as the candidate facility locations in Mechanism 1. In the following we show that any deterministic mechanism which only selects one of endpoints as the facility location cannot do better.

Theorem 2. *Let $N = \{1, 2, \cdots, n\}$, $n \geq 2$. Any deterministic strategy-proof mechanism f which only selects one of the endpoints as the facility location has an approximation ratio of at least 3.*

Proof. Assume $N = \{1, 2\}$ and agents are located on interval $[0,2]$. Let f be a deterministic mechanism which only locates the facility at endpoints. Consider the profile $\mathbf{x} = (x_1, x_2) = (1-\epsilon, 1+\epsilon), 1 > \epsilon > 0$. We have the facility location $y = f(\mathbf{x}) \in \{0, 2\}$. Without loss of generality, suppose that $y = 2$. Thus $cost(y, x_2) = 1 - \epsilon$.

Now consider the profile $\mathbf{x}'$ where $x_1 = 1 - \epsilon$ and $x'_2 = 2$ with new facility location $y' = f(\mathbf{x}')$. By strategy-proofness, the cost from agent 2 must satisfy $cost(y', x_2) \leq 1 - \epsilon$. Otherwise agent 2 gains from deviating from x_2 to x'_2. Therefore $y' = 2$. Hence, for the profile $\mathbf{x}' = (1 - \epsilon, 2)$, $\text{OPT}(\mathbf{x}') = 3 - \epsilon$ and $SC(f, \mathbf{x}') = 1 + \epsilon$. It follows that the approximation ratio of f is $\frac{3-\epsilon}{1+\epsilon}$ which approaches to 3 when ϵ tends to 0. □

If the candidate facility location is not restricted to the endpoints, the next theorem establishes a lower bound of 2 for the strategy-proof deterministic mechanism.

Theorem 3. *Let $N = \{1, 2, \cdots, n\}$, $n \geq 2$. Any deterministic strategy-proof mechanism f has an approximation ratio of at least 2.*

Proof. Assume $N = \{1, 2\}$ and agents are located on interval $[0, 2]$. Let f be a deterministic mechanism. Consider the profile $\mathbf{x} = (x_1, x_2) = (\frac{2}{3}, \frac{4}{3})$. We denote $y = f(\mathbf{x})$ to be the facility location. Three cases should be discussed with respect to the location of y.

Case 1. $y \in [\frac{2}{3}, \frac{4}{3}]$. Thus the obnoxious social cost is $SC(f, \mathbf{x}) = \frac{2}{3}$, whereas the optimum has a cost of 2. It follows that the approximation ratio of f is 3.

Case 2. $y \in (\frac{4}{3}, 2]$. Then the cost of agent 2 is $cost(y, x_2) = y - \frac{4}{3}$. Now consider the profile $\mathbf{x}'$ where $x_1 = \frac{2}{3}$ and $x_2' = 2$ with new facility location $y' = f(\mathbf{x}')$. By the strategy-proofness, the cost from agent 2 must be at most $y - \frac{4}{3}$, i.e. $cost(y', x_2) \leq y - \frac{4}{3}$. Otherwise agent 2 can benefit by misreporting from x_2 to x_2'. It implies that $2x_2 - y \leq y' \leq y$. Since $y \in (\frac{4}{3}, 2]$ and $x_2 = \frac{4}{3}$, we have $y' \in [\frac{2}{3}, 2]$. It is easy to get that $SC(f, \mathbf{x}') = \frac{4}{3}$ and the optimum value of profile $\mathbf{x}'$ is $\frac{8}{3}$. Hence, the mechanism ratio is at least 2.

Case 3. $y \in [0, \frac{2}{3})$. This case is similar to Case 2. □

4 Randomized Mechanisms

In this section, we consider several randomized mechanisms for the obnoxious facility game. Given a profile $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ on interval $[0, 2]$. By the nice property that one of endpoints must be an optimal solution, we can devise a randomized mechanism directly, that is we return the facility location $y = 0$ and $y = 2$ with probability $\frac{1}{2}$ respectively. It is easy to prove that this mechanism is group strategy-proof and its approximation ratio is at most 2. Furthermore, we propose another two randomized mechanisms which improve the approximation ratio.

Similarly, we denote the number of agents located on subinterval $[0, 1]$ and $(1, 2]$ by n_1 and n_2, respectively. The location y of the facility is decided by the following mechanisms.

Mechanism 2

1. If $n_1 < n_2$, then return $y = 0$ and $y = 2$ with probability $\frac{3}{5}$ and $\frac{2}{5}$, respectively;
2. If $n_1 = n_2$, then return $y = 0$ and $y = 2$ with probability $\frac{1}{2}$, respectively;
3. $n_1 > n_2$, then return $y = 0$ and $y = 2$ with probability $\frac{2}{5}$ and $\frac{3}{5}$, respectively.

Theorem 4. *Mechanism 2 is a group strategyproof $\frac{5}{3}$-approximation mechanism for the obnoxious facility game when the agents are located on a path.*

Proof. In order to prove the group strategy-proofness of Mechanism 2, we must demonstrate that for any coalition $S \subseteq N$, all agents in S cannot gain simultaneously. Here we only show the correctness of the case that $n_1 < n_2$. The proofs for other cases are similar. Suppose that agent i is on $(1, 2]$. Since $n_1 < n_2$, by Mechanism 2 we have

$$cost(f(\mathbf{x}), x_i) = \frac{3}{5}x_i + \frac{2}{5}(2 - x_i) = \frac{4}{5} + \frac{1}{5}x_i > \frac{4}{5} + \frac{1}{5} = 1. \tag{1}$$

Consider profile $\mathbf{x}'$, in which for every $i \notin S$, $x'_i = x_i$. Similarly, we denote n'_1 and n'_2 to be the number of agents on corresponding subintervals after deviating. Clearly, if $n'_1 < n'_2$, then the facility distribution does not change. Thus $cost(f(\mathbf{x}), x_i) = cost(f(\mathbf{x}'), x_i)$, for any agent $i \in N$. But if $n'_1 = n'_2$, then at least one agent in S with $x_i \in (1, 2]$ misreports her location to $x'_i \in [0, 1]$. Therefore by (1),

$$cost(f(\mathbf{x}'), x_i) = \frac{1}{2}x_i + \frac{1}{2}(2 - x_i) = 1 < cost(f(\mathbf{x}), x_i).$$

Similarly, if $n'_1 > n'_2$, then there must be at least one agent $i \in S$ on $(1, 2]$ deviating her location to $[0, 1]$. Combining equation (1),

$$cost(f(\mathbf{x}'), x_i) = \frac{2}{5}x_i + \frac{3}{5}(2 - x_i) = \frac{6}{5} - \frac{1}{5}x_i < 1 < cost(f(\mathbf{x}), x_i).$$

We now turn to showing the approximation ratio of $\frac{5}{3}$ for Mechanism 2. If $n_1 = n_2$, then the obnoxious social cost of the mechanism is

$$SC(f, \mathbf{x}) = \sum_{i=1}^{n} cost(f(\mathbf{x}), x_i) = \frac{1}{2}\sum_{i=1}^{n} x_i + \frac{1}{2}\sum_{i=1}^{n}(2 - x_i) = n.$$

Since $n_1 = n_2$ and $\text{OPT}(\mathbf{x}) = \max\{\sum_{i=1}^{n} x_i, \sum_{i=1}^{n}(2 - x_i)\}$, we have

$$\sum_{i=1}^{n} x_i = \sum_{x_i \in [0,1]} x_i + \sum_{x_i \in (1,2]} x_i \le n_1 + 2n_2 = \frac{3}{2}n;$$

$$\sum_{i=1}^{n}(2 - x_i) = \sum_{x_i \in [0,1]} (2 - x_i) + \sum_{x_i \in (1,2]} (2 - x_i) \le n_2 + 2n_1 = \frac{3}{2}n.$$

Combining above two inequalities, the optimal cost satisfies $\text{OPT}(\mathbf{x}) \le \frac{3}{2}n < \frac{5}{3}SC(f, \mathbf{x})$.

For the case that $n_1 \neq n_2$, we only discuss the approximation ratio when $n_1 > n_2$. The case of $n_1 < n_2$ is similar. If $\text{OPT}(\mathbf{x}) = \sum_{i=1}^{n} x_i$, then

$$\text{OPT}(\mathbf{x}) = \sum_{i=1}^{n} x_i \le n_1 + 2n_2 = n + n_2 < \frac{3}{2}n.$$

The last inequality is from the condition $n_1 > n_2$. And the obnoxious social cost is

$$\begin{aligned} SC(f, \mathbf{x}) &= \frac{2}{5}\sum_{i=1}^{n} x_i + \frac{3}{5}\sum_{i=1}^{n}(2 - x_i) \\ &= \frac{6}{5}n - \frac{1}{5}\sum_{i=1}^{n} x_i \\ &> \frac{6}{5} \times \frac{2}{3}\mathrm{OPT}(\mathbf{x}) - \frac{1}{5}\mathrm{OPT}(\mathbf{x}) \\ &= \frac{3}{5}\mathrm{OPT}(\mathbf{x}). \end{aligned}$$

For the other case that $\mathrm{OPT}(\mathbf{x}) = \sum_{i=1}^{n}(2 - x_i)$,

$$\mathrm{OPT}(\mathbf{x}) = \sum_{i=1}^{n}(2 - x_i) \le 2n_1 + n_2 \le 2n.$$

The obnoxious social cost is

$$\begin{aligned} SC(f, \mathbf{x}) &= \frac{2}{5}\sum_{i=1}^{n} x_i + \frac{3}{5}\sum_{i=1}^{n}(2 - x_i) \\ &= \frac{4}{5}n + \frac{1}{5}\sum_{i=1}^{n}(2 - x_i) \\ &\ge \frac{4}{5} \times \frac{1}{2}\mathrm{OPT}(\mathbf{x}) + \frac{1}{5}\mathrm{OPT}(\mathbf{x}) \\ &= \frac{3}{5}\mathrm{OPT}(\mathbf{x}). \end{aligned}$$

□

It is worth noting that this approximation ratio of $\frac{3}{5}$ is tight for Mechanism 2. Consider the location profile $\mathbf{x} = (0, \cdots, 0)$, where all the agents are located on the left endpoint. Thus the optimal cost is $\mathrm{OPT}(\mathbf{x}) = \sum_{i=1}^{n}(2 - x_i) = 2n$. Since $n_1 = n > n_2 = 0$, by Mechanism 2 we have

$$SC(f, \mathbf{x}) = \frac{2}{5}\sum_{i=1}^{n} x_i + \frac{3}{5}\sum_{i=1}^{n}(2 - x_i) = \frac{6}{5}n = \frac{3}{5}\mathrm{OPT}(\mathbf{x}).$$

In Mechanism 2 we only care which between n_1 and n_2 is larger, and thus the distribution of facility location has nothing to do with the exact values of n_1 and n_2. To improve this mechanism we will take advantage of full information about n_1 and n_2.

Mechanism 3. For a location profile $\mathbf{x}$ on interval $[0, 2]$, return facility location $y = 0$ with probability α and $y = 2$ with probability $(1 - \alpha)$, where $\alpha = \frac{2n_1n_2 + n_2^2}{n_1^2 + n_2^2 + 4n_1n_2}$.

Theorem 5. *Mechanism 3 is a group strategyproof $\frac{3}{2}$-approximation mechanism for the obnoxious facility game when the agents are located on a path.*

Proof. Given a location profile $\mathbf{x}$ on interval [0,2]. We first consider two special cases that $n_1 = 0$ or $n_2 = 0$. Clearly, if $n_1 = 0$ or $n_2 = 0$, Mechanism 3 returns the facility location $y = 0$ or $y = 2$ with probability 1. No agent would like to misreport her location. Furthermore when $n_1 = 0$, the obnoxious social cost is equal to the optimal cost, that is $SC(f, \mathbf{x}) = \mathrm{OPT}(\mathbf{x}) = \sum_{i=1}^{n} x_i$. Similarly when $n_2 = 0$, $SC(f, \mathbf{x}) = \mathrm{OPT}(\mathbf{x}) = \sum_{i=1}^{n}(2 - x_i)$.

Next, we prove the strategy-proofness and the approximation ratio of Mechanism 3 if $n_1 \neq 0$ and $n_2 \neq 0$. Here we introduce a parameter $\beta = \frac{n_1}{n_2}$, $(\infty > \beta > 0)$. So α can be rewritten as a function of β, i.e. $\alpha(\beta) = \frac{2\beta+1}{\beta^2+4\beta+1}$. It is obvious that

$$\alpha'(\beta) = -\frac{2(\beta^2 + \beta + 1)}{(\beta^2 + 4\beta + 1)^2} < 0,$$

which implies that $\alpha(\beta)$ monotonously decreases on β.

In order to prove the group strategy-proofness, we denote $S \subseteq N$ to be a coalition. If new agent number n_1' and n_2' satisfies that $\frac{n_1'}{n_2'} = \frac{n_1}{n_2}$, i.e. $\beta' = \beta$, then $cost(f(\mathbf{x}'), x_i) = cost(f(\mathbf{x}), x_i)$ for any $i \in N$. But if $\frac{n_1'}{n_2'} > \frac{n_1}{n_2}$, then there at least one agent on $(1, 2]$ misreports her location to $[0, 1]$. By the fact that $\alpha(\beta)$ monotonously decreases on β, we have $\alpha(\beta) > \alpha(\beta')$. Furthermore,

$$\begin{aligned} cost(f(\mathbf{x}), x_i) - cost(f(\mathbf{x}'), x_i) &= \big[\alpha(\beta)x_i + (1 - \alpha(\beta))(2 - x_i)\big] - \\ &\quad \big[\alpha(\beta')x_i + (1 - \alpha(\beta'))(2 - x_i)\big] \\ &= 2(x_i - 1)\big(\alpha(\beta) - \alpha(\beta')\big) > 0. \end{aligned}$$

Similarly, if $\frac{n_1'}{n_2'} < \frac{n_1}{n_2}$, then $\alpha(\beta) < \alpha(\beta')$ and at least one agent on $[0, 1]$ deviate her location to $x_i' \in (1, 2]$. Hence,

$$cost(f(\mathbf{x}), x_i) - cost(f(\mathbf{x}'), x_i) = 2(x_i - 1)\big(\alpha(\beta) - \alpha(\beta')\big) \geq 0.$$

Let us now turn to prove the $\frac{3}{2}$-approximation ratio of Mechanism 3. We consider two cases by distinguishing the optimal cost.

Case 1. $\mathrm{OPT}(\mathbf{x}) = \sum_{i=1}^{n} x_i$. Hence, $\mathrm{OPT}(\mathbf{x}) \leq n_1 + 2n_2 = \frac{2+\beta}{1+\beta}n$. The obnoxious social cost is

$$\begin{aligned} SC(f, \mathbf{x}) &= \alpha(\beta)\sum_{i=1}^{n} x_i + \big(1 - \alpha(\beta)\big)\sum_{i=1}^{n}(2 - x_i) \\ &\geq \frac{2\alpha(\beta) + \beta}{2 + \beta}\mathrm{OPT}(\mathbf{x}) \\ &= \frac{\beta^2 + 2\beta + 1}{\beta^2 + 4\beta + 1}\mathrm{OPT}(\mathbf{x}) \\ &\geq \frac{2}{3}\mathrm{OPT}(\mathbf{x}). \end{aligned} \tag{2}$$

Case 2. $\mathrm{OPT}(\mathbf{x}) = \sum_{i=1}^{n}(2 - x_i)$. Hence, $\mathrm{OPT}(\mathbf{x}) \leq n_2 + 2n_1 = \frac{1+2\beta}{1+\beta}n$. The obnoxious social cost is

$$\begin{aligned} SC(f, \mathbf{x}) &= \alpha(\beta)\sum_{i=1}^{n} x_i + \big(1 - \alpha(\beta)\big)\sum_{i=1}^{n}(2 - x_i) \\ &\geq \frac{1 + 2\beta - 2\alpha(\beta)\beta}{1 + 2\beta}\mathrm{OPT}(\mathbf{x}) \\ &= \frac{\beta^2 + 2\beta + 1}{\beta^2 + 4\beta + 1}\mathrm{OPT}(\mathbf{x}) \\ &\geq \frac{2}{3}\mathrm{OPT}(\mathbf{x}). \end{aligned} \tag{3}$$

To prove the last inequality of (2) and (3), we denote function $h(\beta) = \frac{\beta^2+2\beta+1}{\beta^2+4\beta+1}$. Then $h'(\beta) = 2\frac{\beta^2-1}{(\beta^2+4\beta+1)^2}$. It is easy to know that $h(\beta)$ monotonously decreases if $0 \leq \beta < 1$ and monotonously increases if $\beta > 1$. Hence, $h(\beta)$ achieves its minimum when $\beta = 1$ and $h_{\min}(\beta) = \frac{2}{3}$. Therefore, we get that the approximation ratio of Mechanism 3 is $\frac{3}{2}$. □

Similarly, we also can show that this approximation ratio of $\frac{3}{2}$ is tight for Mechanism 3. Consider a location profile $\mathbf{x} = (\underbrace{1, \cdots, 1}_{n_1}, \underbrace{2 \cdots, 2}_{n_2})$ and $n_1 = n_2 = \frac{1}{2}n$. It is obvious that $\mathrm{OPT}(\mathbf{x}) = \sum_{i=1}^{n} x_i = \frac{3}{2}n$. By Mechanism 3, we can get $\alpha = \frac{1}{2}$. That is we choose the left and right endpoint with probability $\frac{1}{2}$, respectively. Thus the obnoxious social cost is $SC(f, \mathbf{x}) = \frac{1}{2}\sum_{i=1}^{n} x_i + \frac{1}{2}\sum_{i=1}^{n}(2 - x_i) = n = \frac{2}{3}\mathrm{OPT}(\mathbf{x})$.

5 Concluding Remarks

This paper studies the obnoxious facility game on a path. Our goal is to design deterministic and randomized strategy-proof mechanisms with small approximation ratios. There remain quite a few open problems. One of them is to close the gap between the lower and the upper bounds for the deterministic mechanism. It is interesting to explore the strategy-proof mechanisms for more general networks. For the case that all agents are located on a tree T with vertex set V and edge set E, we can devise a deterministic mechanism similar to Mechanism 1. Based on the location of agents, we construct a new tree graph $T' = (V', E')$, V' is obtained by adding all the locations in $\bigcup_{i=1}^{n} x_i$ which are not in V to V and the edge lengths for the edges that are incident with these new vertices are correspondingly updated. Let $P[a, b]$ be a diameter of T' with two endpoints a and b and point m_{ab} be the midpoint on $P[a, b]$. Suppose that m_{ab} is on edge $[r, s]$ where r is closer to a than to b. If m_{ab} happens to be a vertex, then we assume that it coincides with r. By deleting edge $[r, s]$, T' is decomposed into two subtrees T'_a and T'_b. Note that T'_a contains a and T'_b contains b. Let n_1 and n_2 be the number of agents on subtree T'_a and T'_b. Our mechanism is that if $n_1 \leq n_2$, then the facility location $y = a$; otherwise $y = b$. By using the similar method in proof of Theorem 1, we can prove this mechanism is group strategy-proof and its approximation ratio is at most 3.

References

1. Alon, N., Feldman, M., Procaccia, A.D., Tennenholtz, M.: Strategyproof approximation mechanisms for location on networks. CoRR, abs/0907.2049 (2009)
2. Church, R.L., Garfinkel, R.S.: Locating an obnoxious facility on a network. Transp. Sci. 12, 107–118 (1978)
3. Lu, P., Sun, X., Wang, Y., Zhu, Z.: Asymptotically optimal strategy-proof mechanisms for two-facility games. In: Proceedings of the 11th ACM Conference on Electronic Commerce, (ACM-EC) (2010)
4. Lu, P., Wang, Y., Zhou, Y.: Tighter bounds for facility games. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 137–148. Springer, Heidelberg (2009)
5. Nisan, N.: Introduction to mechanism design (for computer scientists). In: Nisan, N., Roughgarden, T., Tardos, É., Vazirani, V. (eds.) Algorithmic Game Theory. ch.9. Cambridge University Press, Cambridge
6. Procaccia, A.D., Tennenholtz, M.: Approximate mechanism design without money. In: Proceedings of the 10th ACM Conference on Electronic Commerce, (ACM-EC) (2009)
7. Schummer, J., Vohra, R.V.: Mechanism design without money. In: Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V. (eds.) Algorithmic Game Theory. ch.10. Cambridge University Press, Cambridge (2007)
8. Tamir, A.: Obnoxious facility location on graphs. SIAM Journal on Discrete Mathematics 4, 550–567 (1991)
9. Ting, S.S.: A linear-time algorithm for maxisum facility location on tree networks. Transp. Sci. 18, 76–84 (1984)
10. Zelinka, B.: Medians and peripherians of trees. Arch. Math. 4, 87–95 (1968)

Algorithmic Aspects of Heterogeneous Biological Networks Comparison[⋆]

Guillaume Blin[1], Guillaume Fertin[2], Hafedh Mohamed-Babou[2], Irena Rusu[2], Florian Sikora[1], and Stéphane Vialette[1]

[1] Université Paris-Est, LIGM - UMR CNRS 8049, France
{Guillaume.Blin,Florian.Sikora,Stephane.Vialette}@univ-mlv.fr
[2] Université de Nantes, LINA - UMR CNRS 6241, France
{Guillaume.Fertin,Hafedh.Mohamed-Babou,Irena.Rusu}@univ-nantes.fr

Abstract. Biological networks are commonly used to model molecular activity within the cell. Recent experimental studies have shown that the detection of conserved subnetworks across several networks, coming from different organisms, may allow the discovery of disease pathways and prediction of protein functions. There already exist automatic methods that allow to search for conserved subnetworks using networks alignment; unfortunately, these methods are limited to networks of same type, thus having the same graph representation. Towards overcoming this limitation, a unified framework for pairwise comparison and analysis of networks with different graph representations (in particular, a directed acyclic graph D and an undirected graph G over the same set of vertices) is presented in [4]. We consider here a related problem called k-DAGCC: given a directed graph D and an undirected graph G on the same set V of vertices, and an integer k, does there exist sets of vertices $V_1, V_2, \dots V_{k'}$, $k' \leq k$ such that, for each $1 \leq i \leq k'$, (i) $D[V_i]$ is a DAG and (ii) $G[V_i]$ is connected ? Two variants of k-DAGCC are of interest: (a) the V_is must form a *partition* of V, or (b) the V_is must form a *cover* of V. We study the computational complexity of both variants of k-DAGCC and, depending on the constraints imposed on the input, provide several polynomial-time algorithms, hardness and inapproximability results.

1 Introduction

Recent advances in the field of biological networks comparison, inspired by developments in sequence comparison, have provided a prominent tool for explaining organization and interpreting function and evolution of biological networks [20]. In their most basic abstraction level, biological networks can be modeled by graphs, where vertices represent cellular compounds, and edges (or arcs) represent their interactions. These graphs may be directed or undirected, depending on the type of networks they represent: for instance, a Protein-Protein Interaction network (PPI) is usually modeled by an undirected graph whose vertices

[⋆] This work was partially supported by GDR-IM and ANR project BIRDS JCJC SIMI 2-2010.

are proteins and edges represent the physical interactions between proteins; however, a metabolic network is modeled by a directed graph (also called *reaction graph*), where (a) any vertex represents a reaction which needs both a set of input compounds (substrates) and some enzymes (proteins) as catalysts, and produces a set of compounds (products) and (b) there is an arc from reaction r_i to reaction r_j if r_j uses, as substrate, a product of r_i.

Each biological network represents a partial view of the molecular activity within the cell. Thus, comparing them provides a better understanding of the behavior of a biologic system. We say that two biological networks are *homogeneous* if they are of the same type and have the same graph representation, but come from different species. In contrast, networks are said to be *heterogeneous* if they come from the same species, but are of different types and have different graph representations (e.g., a directed graph and an undirected graph).

The main comparative approaches considered so far focus on homogeneous networks using *networks alignment* (see, among others, [10,11,18,20,5,2,21,12]). Such methods are powerful for detecting conserved modules across several networks of different species. Unlike homogeneous networks, only few research works aim at comparing heterogeneous networks. Most of the existing methods used to compare heterogeneous networks are usually manual, or use case-by-case methods. In [22,19,13,16,1], authors search for a chain of reactions in a metabolic network, by investigating the relationships between genes involved in the reactions and the proximity of genes along the genome. Interactions between proteins that catalyze successive reactions in metabolic networks have been studied in [3,9]. In [14,8,6], authors search signaling pathways by confronting a signal transduction network, represented by a directed graph, to a PPI network, represented by an undirected graph. They perform an orientation of the PPI graph as follows: given a list of ordered source-target pairs, the PPI graph is oriented in such a way that, for a maximum number of pairs (s, t), there is a directed path from the source s to the target t. The advantage of this method is that it allows to return to homogeneous networks with the same graph. However, in addition to its computational difficulty, the good behavior of this method is closely related to the choice of the list of the source-target pairs. In [8], such a list is manually constructed.

Recently, a unified framework for comparison and analysis of heterogeneous networks was proposed [4]. In particular, the central problem considered in [4] is the following: given a directed acyclic graph (DAG) D' and an undirected graph G' built on the same set of vertices, find the longest directed path in D' whose vertices induce a connected component in G'. In this setting, D' and G' represent a metabolic network and a PPI network, respectively. However, in general, metabolic networks are not DAGs, and hence contain directed cycles.

Therefore, we consider in thispaper the following problem, called k-DAGCC, that can be seen as a way to pre-process the input graphs in [4]: given a directed graph D and an undirected graph G on the same set V of vertices and an integer k, does there exist sets of vertices $V_1 \dots V_{k'}$, $k' \leq k$, such that, for each $1 \leq i \leq k'$, (i) $D[V_i]$ is a DAG and (ii) $G[V_i]$ is connected ? Two variants of k-DAGCC are considered: (a) the *partition* version, where the V_is must form a partition of V, and (b) the *cover* version, where the V_is must form a cover of V.

The problem k-DAGCC (under both its variants) is also motivated by the research of pathways in metabolic networks that correspond to functionally related proteins in PPI networks. Indeed, pathways in metabolic networks correspond to DAGs [18], while functionally related proteins in PPI networks correspond to connected subgraphs [1,2,15,21,12]). Thus, it is of interest to be able to extract biologically relevant information from two large networks by decomposing them into smaller modules that each (i) carry a rich biological information and (ii) are easier to interpret.

In this paper, we study the computational complexity of both variants of k-DAGCC and, depending on the constraints imposed on the input, provide several polynomial-time algorithms, along with hardness and inapproximability results. Once the main notations and definitions will be stated (Section 2), we will define formally and study in detail the complexity of the two variants of the k-DAGCC problem, first in its partition version (Section 3), then in its cover version (Section 4).

2 Preliminaries

This paper is concerned with both directed and undirected graphs. We briefly recall the basic needed material.

A *graph* $G = (V, E)$ consists of a set of vertices V and a set of edges (unordered pairs of vertices) E. To shorten the exposition, we shall usually abbreviate $|V|$ and $|E|$ to n and m, respectively. The *degree* of a vertex $u \in V$ is the number of edges incident to it. A graph is *acyclic* if it does not contain any cycle. An *independent set* is a subset $V' \subseteq V$ such that $(x, y) \notin E$ for any $x, y \in V'$. A graph is *connected* if there exists a path between any pair of vertices. For any $V' \subseteq V$, we let $G[V'] = (V', E')$ stand for the *subgraph induced* by V', i.e. $E' = \{\{x, y\} \in E : x, y \in V'\}$. A graph is a tree if it is both connected and acyclic. A *star* of order $n \geq 3$ is a tree with exactly one vertex of degree strictly greater than 1. A graph is *complete* if it contains all possible edges. A *planar* graph is a graph that can be embedded onto the plane, in such a way that its edges intersect only at their endpoints. Finally, a graph is said to be *outerplanar* if it has a planar embedding such that the vertices lie on a circle and the edges lie inside that circle, without crossing each other.

A *directed graph* $D = (V, A)$ consists of a set of vertices V and a set of arcs (ordered pairs of vertices) A. A *directed acyclic graph* (DAG) is a directed graph that does not have any directed cycle. For any $V' \subseteq V$, we let $D[V'] = (V', A')$ stand for the *subgraph induced* by V', i.e. $A' = \{(x, y) \in A : x, y \in V'\}$.

We shall consider graphs and directed graphs defined on the same set of vertices, i.e. we will write $D = (V, A)$ for the directed graph and $G = (V, E)$ for the undirected graph. To avoid ambiguity, we let m_D and m_G stand for $|A|$ and $|E|$, respectively. Given two such graphs $D = (V, A)$ and $G = (V, E)$ built on the same set V of vertices, we say that a partition (resp. a cover) $\{V_1, V_2 \ldots V_k\}$ of

V is *valid* if, for any $1 \leq i \leq k$, $D[V_i]$ is a DAG and $G[V_i]$ is connected. The two variants of k-DAGCC we are going to study in this paper are called respectively k-DAGCC-Partition and k-DAGCC-Cover, and are defined as follows.

k-DAGCC-Partition
Instance: A directed graph $D = (V, A)$, an undirected graph $G = (V, E)$, and an integer k.
Question: Does there exist a valid partition $\mathcal{P} = \{V_1, V_2, \ldots, V_{k'}\}$ of V such that $k' \leq k$?

k-DAGCC-Cover
Instance: A directed graph $D = (V, A)$, an undirected graph $G = (V, E)$, and an integer k.
Question: Does there exist a valid cover $\mathcal{C} = \{V_1, V_2, \ldots, V_{k'}\}$ of V such that $k' \leq k$?

The natural minimization version of the above decision problems are denoted Min-DAGCC-Partition and Min-DAGCC-Cover, respectively.

Table 1. Complexity results for k-DAGCC-Partition and Min-DAGCC-Partition

k \ G	Graph	Outerplanar	Tree	Star	Path
$(n-k) = \mathcal{O}(1)$	P [Prop. 2]				
$k = \mathcal{O}(1)$	NPC [Prop. 4]	P [Prop. 3]			
k unbounded	Inapprox. within $n^{1-\epsilon}$ [Prop. 6]	APX-hard [Prop. 7]		NPC [Prop. 5]	P [Prop. 1]

Table 2. Complexity results for k-DAGCC-Cover and Min-DAGCC-Cover

k \ G	Graph	Outerplanar	Tree	Star	Path
$k = \mathcal{O}(1)$	NPC [Prop. 9]				P [Prop. 8]
k unbounded	Inapprox. within $n^{1-\epsilon}$ [Prop. 10]				P [Prop. 8]

The results presented in this paper are summarized in Table 1 (for problems k-DAGCC-Partition and Min-DAGCC-Partition) and Table 2 (for problems k-DAGCC-Cover and Min-DAGCC-Cover).

3 Partition Version of k-DAGCC

In this section, we provide polynomial-time algorithms, hardness and inapproximability results for different variants of Min-DAGCC-Partition and k-DAGCC-Partition (see Table 1). We first provide polynomial-time algorithms for three restricted cases: (a) G is a path, (b) $n-k$ is a constant, and (c) G is an outerplanar graph and k is a constant.

Recall that, given a directed graph $D=(V,A)$ and an undirected graph $G=(V,E)$, testing whether D is a DAG (resp. whether G is connected) can be done in $O(n+m_D)$ time (resp. $O(n+m_G)$ time) by depth-first search.

Proposition 1. Min-DAGCC-Partition *is polynomial-time solvable when G is a path.*

Proof. The proof is by a simple greedy algorithm. Write $(v_1, v_2, \ldots, v_n)$ for the path G. We consider the vertices of G from v_1 to v_n. We start with $S=\emptyset$. For vertex v_i, if $D[S\cup\{v_i\}]$ is a DAG, we add v_i to S, otherwise we report S as an element of the sought partition and set $S=\{v_i\}$. When the algorithm stops, we report the current S as the last element of the partition.

It is easily seen that this greedy algorithm produces a valid partition of V. What is left is to prove that the valid partition obtained by the above algorithm is of minimum cardinality. Let $\mathcal{P}=\{V_1, V_2, \ldots V_k\}$ be the partition returned by our algorithm, and suppose, aiming at a contradiction, that there exists a strictly smaller valid partition $\mathcal{P}'=\{V_1', V_2', \ldots, V_\ell'\}$, thus with $\ell<k$. We first observe that, by construction, V_1 cannot be a proper subset of V_1'. Since any connected subgraph of G is built on consecutively indexed vertices $\{v_p, v_{p+1}, v_{p+2}\ldots v_q\}$, it follows that there must exist $2\leq i<k$ and $2\leq j\leq\ell$ such that (1) V_i is a proper subset of V_j', and (2) V_j' contains the first vertex, say x, of V_{i+1}. This is a contradiction since $V_i\cup\{x\}$ is not a DAG in D. $\square$

Proposition 2. k-DAGCC-Partition *is polynomial-time solvable when $n-k$ is a constant.*

Proof. Let us prove that this variant of k-DAGCC-Partition can be solved in polynomial time by an exhaustive procedure. We consider all k-partitions of V. We start from the unique n-partition of V and iteratively compute all $(k-1)$-partitions of V starting from its k-partitions. Recall that the Stirling number of the second kind $S(n,k)$ represents the number of ways to partition a set of n elements into k nonempty subsets. We have $S(n,k)=S(n,n)\prod_{i=0}^{n-k-1}\binom{n-i}{2}$. But $S(n,n)=1$, and hence $S(n,k)=\prod_{i=0}^{n-k-1}\binom{n-i}{2}=O(n^{2(n-k)})$. For each of these $O(n^{2(n-k)})$ partitions, we can check in polynomial-time whether it is a valid solution for k-DAGCC-Partition. The proposition follows. $\square$

Proposition 3. k-DAGCC-Partition *is polynomial-time solvable when G is an outerplanar graph and k is a constant.*

Proof. Recall that a graph is outerplanar just when its vertices can be placed on a circle so that its edges become non-crossing chords of the circle. Fix any such an outerplanar circular embedding. Let $V_1, V_2, \ldots, V_k$ be any valid partition of V. Now, associate to any V_i, $1 \leq i \leq k$, the smallest arc of the circle that covers all vertices in V_i (according to the outerplanar circular embedding). Since $V_1, V_2, \ldots, V_k$ is a valid partition of V, any vertex is covered by at least one arc of the circle, and no two arcs of the circle share a common endpoint. Moreover, since $(v_1, v_2, \ldots, v_n)$ is an outerplanar embedding of G and $G[V_i]$ is connected for every $1 \leq i \leq k$, it follows that no two arcs of the circle properly overlap.

The algorithm is by enumerating all sets of non-overlapping k arcs of the circle with distinct endpoints that cover V. Let $v_1, v_2, \ldots, v_n$ be the vertices of G in the outerplanar circular embedding following, say, the trigonometric orientation. For any $1 \leq i \leq n$, consider all $2(k-1)$-subsets of $V \setminus \{v_i, v_{i-1}\}$ (by convention, $v_0 = v_n$). Let $\{v_{j_1}, v_{j_2}, \ldots, v_{j_{2(k-1)}}\}$ be such a subset, where the vertices are read along the outerplanar circular embedding starting from v_i and following the trigonometric orientation. We now need to construct all possible arcs of the circle with endpoints $v_i, v_{j_1}, v_{j_2}, \ldots, v_{j_{2(k-1)}}, v_{i-1}$ such that no two arcs of the circle are overlapping. Clearly, this reduces to considering all well-formed parenthesis strings of length $2k$. Our construction yields sets of k arcs $a_1, a_2, \ldots, a_k$ of the circle such that (1) any two arcs of the circle have distinct endpoints, (2) no two arcs of the circle are overlapping, and (3) each vertex of V is covered by at least one arc of the circle. For any such solution, we construct a partition $(V_1, V_2, \ldots, V_k)$ as follows: V_i contains all vertices covered by arc a_i of the circle, except those vertices covered by at least one arc a_j of the circle which is strictly covered by a_i. This algorithm is $O(n^2 \binom{n}{2(k-1)} \binom{2(k-1)}{k-1} (n+m))$ time, where $m = \max\{m_D, m_G\}$. Indeed, we have n possibilities for choosing the reference vertex v_i. Starting from each reference vertex v_i, we consider $\binom{n}{2(k-1)} \frac{1}{k} \binom{2(k-1)}{k-1}$ sets of k arcs of the circle ($\frac{1}{k} \binom{2(k-1)}{k-1}$ is the number of well-formed parenthesis strings of length $2k$). From these sets, one can construct a solution V_i, $1 \leq i \leq k$, is $O(n)$ time, and check whether each subset V_i, $1 \leq i \leq k$, induces both a DAG in D and a connected component in G in $O(n+m)$ time. The proposition follows. □

We now prove that k-DAGCC-Partition becomes NP-complete when (a) G is a complete graph or (b) G is a star.

Proposition 4. *For any $k \geq 2$, k-DAGCC-Partition is NP-complete even when G is a complete graph.*

Proof. k-DAGCC-Partition is certainly in NP. To prove hardness, we propose a reduction from the NP-complete Not-All-Equal 3SAT (NAE 3Sat) problem [7]: Given a collection $\mathcal{C}_q = \{c_1, \ldots c_q\}$ of q clauses, where each clause consists of a set of three literals over a finite set of n boolean variables $\mathcal{V}_n = \{x_1, \ldots x_n\}$, is there a truth assignment of each variable of $\mathcal{V}_n$ such that at least one of the literals is `true` and at least one is `false` in each clause ? For ease of exposition, for any $1 \leq j \leq 3$, we let c_i^j stand for the j-th literal of clause c_i.

Given any instance $(\mathcal{C}_q, \mathcal{V}_n)$ of NAE 3Sat, we build D and G as follows:

- $V = \{v_i^j : 1 \leq i \leq q, 1 \leq j \leq 3\} \cup \{v_i : 3 \leq i \leq k\}$

- $A = \{(v_i^j, v_{i'}^{j'}), (v_{i'}^{j'}, v_i^j) : c_i^j = \overline{c_{i'}^{j'}}\} \cup \{(v_i^1, v_i^2), (v_i^2, v_i^3), (v_i^3, v_i^1) : 1 \leq i \leq q\} \cup \{(v_i, v), (v, v_i) : 3 \leq i \leq k, v \in V \setminus \{v_i\}\}$
- $E = \{(v, v') : v, v' \in V\}$

Roughly speaking, vertex v_i^j corresponds to the j-th literal of clause c_i, whereas vertices v_i, $3 \leq i \leq k$, are gadgets to adapt the proof for any $k \geq 2$. There is a cycle of length two between any pair of vertices representing two complementary literals (e.g. x_i and $\overline{x_i}$) and between v_i, $3 \leq i \leq k$, and any other vertex of V. Moreover, there is a cycle of length three between the triple of vertices (v_i^1, v_i^2, v_i^3) corresponding to the three literals of any clause c_i. Finally, G is a complete graph on V. An illustration of such a construction – omitting G – is given in Figure 1.

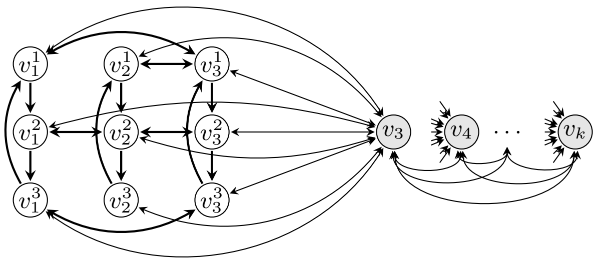


Fig. 1. Illustration of the construction of D, given $\mathcal{C}_q = \{(x_1 \vee x_2 \vee x_3), (x_1 \vee \overline{x_2} \vee x_4), (\overline{x_1} \vee x_2 \vee \overline{x_3})\}$. For readability, arcs $\{(v_i^j, v_l), (v_l, v_i^j) : 1 \leq i \leq q, 1 \leq j \leq 3, 4 \leq l \leq k\}$ are not drawn.

We claim that there is a solution for our NAE 3SAT instance iff there exists a valid partition $(V_1, V_2, \ldots, V_k)$ of V.

($\Rightarrow$) Given a truth assignment $\mathcal{A}$ of $\mathcal{V}_n$ such that each clause contains at least one true and one false literals, we make a partition $\mathcal{P} = \{V_1, V_2 \ldots V_k\}$ of V as follows: $V_1 = \{v_i^j : c_i^j = \texttt{true} \text{ in } \mathcal{A}\}$, $V_2 = \{v_i^j : 1 \leq i \leq q, 1 \leq j \leq 3\} \setminus V_1$, and, $V_i = \{v_i\}$ for $3 \leq i \leq k$. By definition, there is one or two true literal(s) in each clause. Thus, among the three vertices corresponding to the literals of c_i, at least one and at most two of them are in V_1 (resp. V_2). Moreover, considering $\mathcal{A}$, any variable cannot be simultaneously `true` and `false`. Therefore, two vertices representing complementary literals cannot both belong to V_1 (resp. V_2). Then it follows that each $D[V_i]$, $1 \leq i \leq k$, is a DAG since all cycles have been destroyed. Since G is complete, we have a valid solution.

($\Leftarrow$) Let $\mathcal{P} = \{V_1, V_2 \ldots V_k\}$ be a valid partition of V. We build a truth assignment $\mathcal{A}$ as follows. By construction, there is no loss of generality in assuming $V_3 = \{v_3\}, \ldots, V_k = \{v_k\}$ (each of these vertices has to occur as a singleton in the partition). Let us define a truth setting $\mathcal{A}$ such that literal c_i^j, $1 \leq i \leq q$ and $1 \leq j \leq 3$, evaluates to `true` iff $v_i^j \in V_1$. This requirement certainly defines a truth setting assignment since opposite literals are part of a cycle of length two. Let us now prove that $\mathcal{A}$ is a solution for our NAE 3SAT instance. Indeed, at least one and at most two of the vertices representing the literals of each clause c_i belong to V_1. Thus, any clause c_i is indeed satisfied and by at most two of its literals. □

Proposition 5. *k-DAGCC-Partition is NP-complete even when G is a star.*

Proof. We propose a reduction from the NP-complete k'-Independent Set (k'-IS) problem [7]: Given a graph $G_I = (V_I, E_I)$ and a positive integer k', is there an independent set $V_I' \subseteq V_I$ of cardinality at least k' ?

Given any instance G_I of k'-IS, we build D and G as follows:

- $V = V_I \cup \{v_r\}$
- $A = \{(v, v'), (v', v) : (v, v') \in E_I\}$
- $E = \{(v_r, v) : v \in V_I\}$

Moreover, we set $k = |V_I| - k' + 1$. Roughly speaking, graph D is obtained from G_I by replacing each edge by two arcs in opposite directions, and by adding an isolated vertex v_r. The graph G is a star whose center is v_r; thus v_r must be part of any connected subgraph of G of order strictly greater than one. Construction of both these graphs is illustrated in Figure 2.

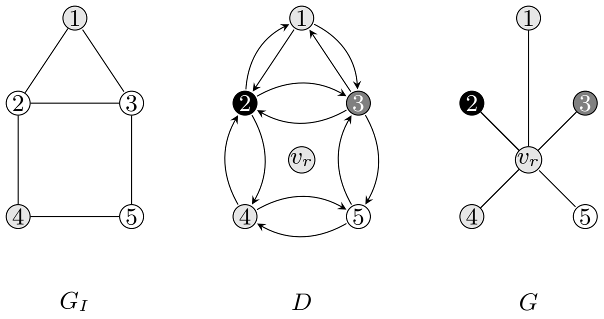


Fig. 2. Illustration of the construction of D and G, given G_I. We highlighted a possible 2-IS $\{1, 4\}$ and a corresponding valid 4-partition of V.

We will show that there is an independent set, in $G_I = (V_I, E_I)$, of cardinality at least k' iff there is a solution for k-DAGCC-Partition with $k = |V_I| - k' + 1$.

($\Rightarrow$) Given an independent set of G_I of cardinality greater or equal than k', choose any subset V_I' of k' vertices of it (which is itself an independent set). We compute the partition $\mathcal{P} = \{V_1, V_2 \dots V_k\}$ of V, where $k = |V_I| - k' + 1$, as follows. Let $V_1 = V_I' \cup \{v_r\}$. Consider any ordering of the vertices in $V_I \setminus V_I'$, and, for every $2 \leq i \leq k$, let $V_i = \{v\}$, where v is the $(i-1)$-th vertex in this ordering. First, note that any $D[V_i]$, $2 \leq i \leq k$ is indeed a DAG since it is composed of a single vertex. Moreover, by definition, $\nexists \{v, v'\} \subseteq V_I'$, such that $(v, v') \in E_I$. Therefore, $D[V_1]$ is composed of isolated vertices (and thus, is also a DAG). The induced subgraphs $G[V_i]$, $2 \leq i \leq k$, are trivially connected; moreover, connectivity of $G[V_1]$ is ensured by the fact that $v_r \in V_1$.

($\Leftarrow$) Given a solution $\mathcal{P} = \{V_1, V_2 \ldots V_k\}$, to k-DAGCC-Partition, we build an independent set $V'_I \subseteq V_I$ of G_I, such that $|V'_I| = |V_I| - k + 1$, as follows. Let $V'_I = V_j \setminus \{v_r\}$ where D_j is the DAG containing v_r. As previously mentioned, in any solution, any connected subgraph of G of order strictly greater than one must contain v_r. Since we require that for every $1 \leq i < j \leq k$, $V_i \cap V_j = \emptyset$, at most one of the V_is can be of order strictly greater than one, and it contains v_r. Thus, $|V'_I| = |V_j| - 1 = ((|V_I| + 1) - (k - 1)) - 1$, that is $|V'_I| = |V_I| - k + 1$. Moreover, V'_I is indeed an independent set since D_j is a DAG and thus does not contain cycles. □

Finally, let us prove the inapproximability of Min-DAGCC-Partition when G is a graph or a tree.

Proposition 6. Min-DAGCC-Partition *cannot be approximated within* $n^{1-\epsilon}$*, for any* $\epsilon > 0$.

Proof. We give an L-reduction from the Minimum Chromatic Number (Min-CN) problem defined as follows: Given a graph $G_C = (V_C, E_C)$, find a proper vertex coloring of G_C using the minimum number of colors, where a vertex coloring is said to be proper iff two neighbors in G_C carry different colors.

Given any instance G_C of Min-CN, we build D and G as follows:

- $V = V_C$
- $A = \{(v, v'), (v', v) : (v, v') \in E_C\}$
- $E = \{(v, v') : v, v' \in V_C\} \setminus E_C$

In other words, we keep the same set of vertices as in G_C, the graph D is obtained from G_C by replacing each edge with two arcs in opposite directions, while G is the complement of G_C. An example of such a construction is given in Figure 3.

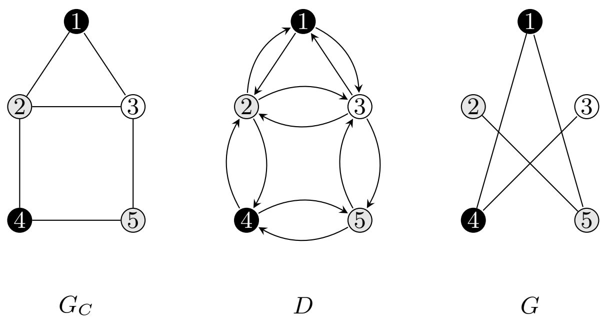


Fig. 3. Illustration of the construction of D and G, given G_C. We highlighted a 3-coloring of D and a corresponding valid 3-partition of V.

Let us prove that this construction is indeed an L-reduction from MIN-CN. More precisely, we will prove the following property: there exists a proper coloring for G_C using k colors iff there exists a valid cardinality k partition $\mathcal{P} = \{V_1, V_2 \dots V_k\}$ of V.

($\Rightarrow$) Given a coloring of G_C with k colors, let V_i, $1 \le i \le k$, be the set of vertices assigned color i. Let us now show that the partition $\mathcal{P} = \{V_i : 1 \le i \le k\}$ is valid (i.e., for every $1 \le i \le k$, $D[V_i]$ is a DAG and $G[V_i]$ is connected). Indeed, by definition of a proper coloring, each V_i, $1 \le i \le k$, is an independent set. Thus, any $D[V_i]$, $1 \le i \le k$, is composed of isolated vertices – and therefore is a DAG. Moreover, the induced subgraphs in G are indeed connected since, by construction, whenever two vertices are not adjacent in G_C, they are in G.

($\Leftarrow$) Given a valid k-partition $P = \{V_1, V_2 \dots V_k\}$ of V, we assign to any vertex $v \in V_i$ the color i, for any $1 \le i \le k$. This assignment is a proper coloring since, by construction, no $D[V_i]$, $1 \le i \le k$, contains an arc, otherwise it would not be a DAG.

The above reduction linearly preserves the approximation, and moreover the sizes of the solutions in the two problems are equal. Hence, given an approximation algorithm for MIN-DAGCC-PARTITION, one can derive an algorithm for MIN-CN with the same approximation ratio. Since MIN-CN cannot be approximated within $n^{1-\epsilon}$ for any $\epsilon > 0$ [23], so does MIN-DAGCC-PARTITION. □

Proposition 7. MIN-DAGCC-PARTITION *is* APX*-hard even when G is a tree.*

Proof. We give an L-reduction from the APX-hard problem SET COVER-2 [17] defined as follows: Given a ground set $\mathcal{X} = \{x_1, \dots x_n\}$ and a collection of sets $\mathcal{C} = \{S_1, \dots S_q\}$ in which each element of $\mathcal{X}$ appears at most twice, find a minimum set cover $\mathcal{C}'$, i.e. a set $\mathcal{C}' \subseteq \mathcal{C}$ such that $\mathcal{X} = \bigcup_{S_i \in \mathcal{C}'} S_i$ and $|\mathcal{C}'|$ is minimum. In the rest of the proof, for any $1 \le i \le q$, we will denote by s_i^j the j-th element of S_i.

Given any instance $(\mathcal{X}, \mathcal{C})$ of SET COVER-2, we build D and G as follows:

- $V = \{v_r\} \cup \{v_i : S_i \in \mathcal{C}\} \cup \{v_i^k : x_k = s_i^j, S_i \in \mathcal{C}, 1 \le j \le |S_i|\}$
- $A = \{(v_i^k, v_j^k), (v_j^k, v_i^k) : x_k \in S_i \cap S_j\} \cup \{(v_r, v_i^k), (v_i^k, v_r) : \nexists S_j \text{ s.t. } x_k \in S_i \cap S_j\}$
- $E = \{(v_r, v_i) : S_i \in \mathcal{C}\} \cup \{(v_i, v_i^k) : x_k \in S_i, S_i \in \mathcal{C}\}$.

Otherwise stated, G is a tree rooted at v_r, whose children are the vertices $v_i, 1 \le i \le q$ (where v_i, $1 \le i \le q$, represents S_i). Each v_i has one child per element of S_i (v_i^k for $x_k \in S_i$). In D, the vertices $v_i, 1 \le i \le q$, are isolated whereas the two vertices (v_i^k, v_j^k) representing an element x_k appearing twice in $\mathcal{C}$ form a cycle of length two. Each of the remaining vertices forms a cycle of length two with v_r. An illustration of such a construction is given in Figure 4.

Let us now prove that this construction is indeed an L-reduction; for this, we show that there is a solution $\mathcal{C}'$ of size k for SET COVER-2 iff there is a valid partition of V in at most $k+1$ sets $V_1, V_2 \dots V_{k+1}$.

($\Rightarrow$) Given a set-cover $\mathcal{C}' \subseteq \mathcal{C}$ of cardinality k, we compute the partition $\mathcal{P} = \{V_1, V_2 \dots, V_k, V_{k+1}\}$ of V as follows: for each $S_i \in \mathcal{C}'$, $1 \le i \le k$, let

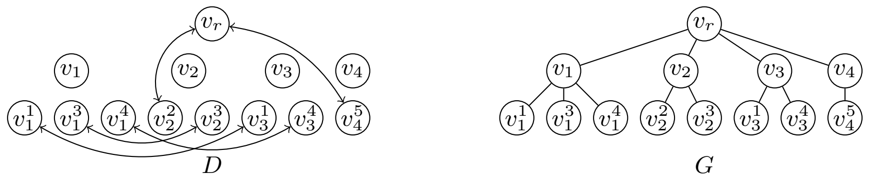

v_r
v_1
v_2
v_3
v_4
v_1^1 v_1^3 v_1^4 v_2^2 v_2^3 v_3^1 v_3^4 v_4^5
D
G

Fig. 4. Illustration of the construction of D and G, given $\mathcal{C} = \{\{x_1, x_3, x_4\}, \{x_2, x_3\}, \{x_1, x_4\}, \{x_5\}\}$

$V_i = \{v_i\} \cup \{v_i^k : x_k \in S_i\}$, and let $V_{k+1} = \{v_r\} \cup \{v_j, v_j^k : x_k \in S_j, S_j \in \mathcal{C} \setminus \mathcal{C}'\}$. In other words, there is a set V_i (inducing the subtree of G rooted at v_i) for each S_i belonging to the set cover; whereas V_{k+1} contains the remaining vertices (including the root v_r of G). Consequently, by construction, for any $1 \leq i \leq k+1$, $G[V_i]$ is indeed connected. Moreover, each $D[V_i]$, $1 \leq i \leq k$, is a DAG since it does not contain v_r and each element of $\mathcal{X}$ occurs at most once in S_i. Finally, $D[V_{k+1}]$ is also a DAG since, by definition of Set Cover-2, any element appears at most twice in $\mathcal{C}$ and at least once in $\mathcal{C}'$. Therefore, in $\mathcal{C} \setminus \mathcal{C}'$, any element appears at most once, and thus no cycle occurs in $D[V_{k+1}]$.

($\Leftarrow$) Given a valid partition $\mathcal{P} = \{V_1, V_2 \dots V_k\}$ of V, we build a cover $\mathcal{C}' = \{S_1, \dots, S_{k-1}\}$ as follows. Assume, wlog, $v_r \in V_1$; then $\mathcal{C}' = \{S_i : \exists v_i^j \notin V_1\}$. In other words, we add in the cover all the sets having an element whose corresponding vertex does not belong to V_1. First, note that, by construction, there is at most one vertex of $\{v_i : 1 \leq i \leq q\}$ in any $G[V_j]$, $2 \leq j \leq k$, since any path linking two such vertices goes through v_r, which is already contained in V_1. It follows that $|\mathcal{C}'| \leq k - 1$. It remains to show that $\mathcal{C}'$ is indeed a set cover for $\mathcal{X}$. To do so, consider any element $x_i \in \mathcal{X}$. If x_i occurs exactly once in $\mathcal{C}$ (wlog, suppose $x_i \in S_j$), then the corresponding vertex v_j^i cannot belong to V_1 since, by construction, it would induce a cycle with v_r in $D[V_1]$. Therefore, $S_j \in \mathcal{C}'$. Otherwise, x_i occurs exactly twice, and then at most one of the corresponding vertices v_j^i and $v_{j'}^i$ belongs to V_1, since otherwise it would induce a cycle in $D[V_1]$. Thus, every $x_i \in \mathcal{X}$ appears at least once in $\mathcal{C}'$.

The above reduction is an L-reduction from Set Cover-2 to Min-DAGCC-Partition. Hence, since Set Cover-2 is APX-hard, so is Min-DAGCC-Partition, and the proposition is proved. □

4 Cover Version of *k*-DAGCC

As we did in the previous section, we now show several complexity results concerning the k-DAGCC-Cover and Min-DAGCC-Cover problems (see also Table 2).

Let us first provide a polynomial-time algorithm when G is a path (Proposition 8). We first prove the following lemma.

Lemma 1. *When G is a path, k-DAGCC-Cover admits a YES answer iff k-DAGCC-Partition admits a YES answer.*

Proof. Any partition being a cover, if k-DAGCC-Partition admits a YES answer, then so does k-DAGCC-Cover. Conversely, consider any positive answer to k-DAGCC-Cover. Hence, there exists a cover $\mathcal{C} = \{V_1, V_2, \ldots V_k\}$ of D such that for every $1 \leq i \leq k$, $G[V_i]$ is connected and $D[V_i]$ is a DAG. Since G is a path, so does any $G[V_i]$. Suppose now that a vertex $v \in V$ belongs to at least three pairwise distinct sets V_p, V_q, V_r from $\mathcal{C}$. In that case, G being a path, one of these three sets, say (wlog) V_p, satisfies $V_p \subseteq V_q \cup V_r$, and we can remove V_p from $\mathcal{C}$ and obtain a strictly smaller cover. Applying this rule until no such case occurs, we end up with a cover $\mathcal{C}' = \{V'_1, \ldots V'_{k'}\}$, $k' \leq k$, in which every vertex $v \in V$ belongs to at most two different sets of $\mathcal{C}'$. Therefore, assuming the V'_is are ordered according to their leftmost vertex in G, we consider the following partition: $\mathcal{P} = \{V''_i = V'_i \setminus V'_{i+1} : 1 \leq i < k' \text{ s.t. } V'_i \in \mathcal{C}'\} \cup \{V'_{k'}\}$. $\mathcal{P}$ is indeed a partition of V, of cardinality $k' \leq k$; besides, (a) since for any $1 \leq i < k'$, $D[V'_i]$ is a DAG, so does any $D[V''_i]$; (b) $D[V'_{k'}]$ is a DAG by hypothesis, and (c) $G[V''_i]$, $1 \leq i < k'$, and $G[V'_{k'}]$ are all subpaths of G, and therefore connected. Thus k-DAGCC-Partition admits a YES answer, and the lemma is proved. □

Proposition 8. Min-DAGCC-Cover *is polynomial-time solvable when G is a path.*

Proof. By Lemma 1, we know that when G is a path, the (YES/NO) solutions of k-DAGCC-Partition and k-DAGCC-Cover are equivalent. Thus, in the minimization versions of both problems (namely, Min-DAGCC-Partition and Min-DAGCC-Cover), the same minimum value is reached, and in particular if there exists a cardinality k solution to Min-DAGCC-Partition, then there exists a cardinality k solution to Min-DAGCC-Cover. Besides, any partition being a cover, we conclude that any solution to Min-DAGCC-Partition is also a solution to Min-DAGCC-Cover Since Min-DAGCC-Partition is polynomial time solvable (see Proposition 1), so does Min-DAGCC-Cover. □

As opposed to k-DAGCC-Partition, k-DAGCC-Cover proves to be NP-complete even when G is a star and k is a constant.

Proposition 9. *For any $k \geq 3$, k-*DAGCC-Cover *is* NP*-complete, even when G is a star.*

Proof. Clearly, k-DAGCC-Cover is in NP. In order to prove that the problem is NP-hard, we provide a reduction from the Minimum Chromatic Number (Min-CN) problem, in its natural decision version: Given a graph $G_C = (V_C, E_C)$ and an integer k, does there exist a proper vertex coloring of G_C using at most k colors ? This problem has been shown to be NP-hard for any $k \geq 3$ [7].

Given any instance G_C of Min-CN, we build D and G as follows:

- $V = V_C \cup \{v_r\}$
- $A = \{(v, v'), (v', v) : (v, v') \in E_C\} \cup \{(v, v_r) : v \in V_C\}$
- $E = \{(v_r, v) : v \in V_C\}$

In other words, D is obtained from G_C by replacing each edge by two arcs in opposite directions, and by adding an arc from any vertex to v_r, while G is a star whose center is v_r.

We now prove the following property: there exists a proper coloring for G_C using k colors iff there exists a valid cardinality k cover $\mathcal{C} = \{V_1, V_2 \ldots V_k\}$ of V.

($\Rightarrow$) Given a proper coloring of G_C with k colors, let S_i, $1 \leq i \leq k$, be the set of vertices assigned color k. We compute a cardinality k cover $\mathcal{C} = \{V_1, V_2 \ldots V_k\}$ of V as follows: for any $1 \leq i \leq k$, $V_i = S_i \cup \{v_r\}$. By definition of a proper coloring, each S_i is an independent set in G_C (thus in D). Hence, since the out-degree of v_r is equal to zero, any $D[V_i]$, $1 \leq i \leq k$, is a DAG. Moreover, any $G[V_i]$ is indeed connected due to the fact that (i) G is star whose center is v_r, and (ii) every V_i, $1 \leq i \leq k$, contains v_r. Hence, $\mathcal{C}$ is valid.

($\Leftarrow$) Given a valid cardinality k cover $\mathcal{C} = \{V_1, V_2 \ldots V_k\}$ of V, we assign to any vertex $v \in V_i \setminus \{v_r\}$ the color i, for any $1 \leq i \leq k$. This assignment is a proper coloring of G_C since, by construction, no V_i can contain two neighbors in G_C, since otherwise, by construction of D, $D[V_i]$ would not be a DAG. $\square$

We finally show that MIN-DAGCC-COVER is hard to approximate. Note that this inapproximability even holds when G is a star (instead of general graphs for the problem MIN-DAGCC-PARTITION).

Proposition 10. MIN-DAGCC-COVER *cannot be approximated within* $n^{1-\epsilon}$*, for any* $\epsilon > 0$*, even when* G *is a star.*

Proof. The reduction provided in proof of Proposition 9 is actually an L-reduction, since the sizes of the solutions in the two problems are equal (we have indeed proved, following our reduction, that "there exists a proper coloring for G_C using k colors iff there exists a valid cardinality k cover $\mathcal{C} = \{V_1, V_2 \ldots V_k\}$ of V"). Hence, given any approximation algorithm for MIN-DAGCC-COVER, one can derive an algorithm for MIN-CN, with the same approximation ratio. Since MIN-CN cannot be approximated within $n^{1-\epsilon}$ for any $\epsilon > 0$ [23], so does MIN-DAGCC-COVER.

5 Conclusion

In this paper, we have studied the problem of decomposing (i.e., partitioning or covering) a directed graph D into DAGs, such that each DAG induces a connected subgraph in a given undirected graph G built on same vertex set as D. We have provided, depending on the constraints imposed on the input, several polynomial-time algorithms, as well as hardness and inapproximability results.

There are still several open problems worthwhile to study. For example, one may consider the parameterized complexity of k-DAGCC-PARTITION, where the parameter is the number of partitions, in the specific case where G is a tree (we indeed know that no FPT algorithm of complexity $f(k)n^{\mathcal{O}(1)}$ is possible when G is a graph, since the problem is NP-complete for $k = 2$). One may also consider studying the approximability of MIN-DAGCC-PARTITION.

References

1. Boyer, F., Morgat, A., Labarre, L., Pothier, J., Viari, A.: Syntons, metabolons and interactons: an exact graph-theoretical approach for exploring neighbourhood between genomic and functional data. Bioinformatics 21(23), 4209–4215 (2005)
2. Deniélou, Y.-P., Boyer, F., Viari, A., Sagot, M.-F.: Multiple alignment of biological networks: A flexible approach. In: Kucherov, G., Ukkonen, E. (eds.) CPM 2009. LNCS, vol. 5577, pp. 263–273. Springer, Heidelberg (2009)
3. Durek, P., Walther, D.: The integrated analysis of metabolic and protein interaction networks reveals novel molecular organizing principles. BMC Syst. Biol. 2(1) (2008)
4. Fertin, G., Babou, H.M., Rusu, I.: A pattern-guided approach to compare heterogeneous networks. Submitted (2011), `http://pagesperso.lina.univ-nantes.fr/~E09D478T/SGM-DB.pdf`
5. Flannick, J., Novak, A., Srinivasan, B.S., McAdams, H.H., Batzoglou, S.: Graemlin: General and robust alignment of multiple large interaction networks. Genome Res. 16(9), 1169–1181 (2006)
6. Gamzu, I., Segev, D., Sharan, R.: Improved orientations of physical networks. In: Moulton, V., Singh, M. (eds.) WABI 2010. LNCS, vol. 6293, pp. 215–225. Springer, Heidelberg (2010)
7. Garey, M., Johnson, D.: Computers and Intractability: A guide to the theory of NP-completeness. W.H. Freeman, San Francisco (1979)
8. Gitter, A., Klein-Seetharaman, J., Gupta, A., Bar-Joseph, Z.: Discovering pathways by orienting edges in protein interaction networks. Nucleic Acids Research 39(4), e22 (2011)
9. Huthmacher, C., Gille, C., Holzhütter, H.: A computational analysis of protein interactions in metabolic networks reveals novel enzyme pairs potentially involved in metabolic channeling. J. Theor. Biol. 252(3), 456–464 (2008)
10. Kelley, B.P., Sharan, R., Karp, R.M., et al.: Conserved pathways within bacteria and yeast as revealed by global protein network alignment. Proc. Natl. Acad. Sci. USA 100(20), 11394–11399 (2003)
11. Kelley, B.P., Yuan, B., Lewitter, F., Sharan, R., Stockwell, B.R., Ideker, T.: Pathblast: a tool for alignment of protein interaction networks. Nucleic Acids Res, 32(Web Server issue) (2004)
12. Kuchaiev, O., Milenkovic, T., Memisevic, V., Hayes, W., Przulj, N.: Topological network alignment uncovers biological function and phylogeny. J. R. Soc. Interface 7(50), 1341–1354 (2010)
13. Lee, I., Date, S.V., Adai, A.T., Marcotte, E.M.: A probabilistic functional network of yeast genes. Science 306, 1555–1558 (2004)
14. Medvedovsky, A., Bafna, V., Zwick, U., Sharan, R.: An algorithm for orienting graphs based on cause-effect pairs and its applications to orienting protein networks. In: Crandall, K.A., Lagergren, J. (eds.) WABI 2008. LNCS (LNBI), vol. 5251, pp. 222–232. Springer, Heidelberg (2008)
15. Narayanan, M., Karp, R.M.: Comparing protein interaction networks via a graph match-and-split algorithm. J of Comput. Biol. 14(7), 892–907 (2007)
16. Pal, C., Hurst, L.: Evidence against the selfish operon theory. Trends Genet. 20, 232–234 (2004)
17. Papadimitriou, C., Yannakakis, M.: Optimization, approximation, and complexity classes. J. Comput. Syst. Sci. 43(3), 425–440 (1991)
18. Pinter, R.Y., Rokhlenko, O., Yeger-Lotem, E., Ziv-Ukelson, M.: Alignment of metabolic pathways. Bioinformatics 21(16), 3401–3408 (2005)

19. Rison, S., Teichmann, S., Thornton, J.: Homology, pathway distance and chromosomal localisation of the small molecule metabolism enzymes in Escherichia coli. J. Mol. Biol. 318, 911–932 (2002)
20. Sharan, R., Ideker, T.: Modeling cellular machinery through biological network comparison. Nature Biotechnol. 4(4), 427–433 (2006)
21. Tian, W., Samatova, N.F.: Pairwise alignment of interaction networks by fast identification of maximal conserved patterns. In: Proc. 14th Pacific Symposium on Biocomputing (PSB), pp. 99–110 (2009)
22. Zheng, Y., Szustakowski, J., Fortnow, L., Roberts, R., Kasif, S.: Computational identification of operons in microbial genomes. Genome Res. 12, 1221–1230 (2002)
23. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory of Computing 3(1), 103–128 (2007)

Minimum Interval Cover and Its Application to Genome Sequencing

Liang Ding[1], Bin Fu[1], and Binhai Zhu[2]

[1] Department of Computer Science University of Texas-Pan American, Edinburg, TX 78541, USA
adamdingliang@gmail.com, binfu@cs.panam.edu

[2] Department of Computer Science Montana State University, Bozeman, MT 59717, USA
bhz@cs.montana.edu

Abstract. Pairwise end sequencing is a very useful method for whole genome sequencing which determines the complete DNA sequence of an organism's genome with the help with laboratory processes. Paired-end interval cover problem is derived from pairwise end sequencing. A paired-end interval for a sequence S is constituted of at most two disjoint intervals, and the paired-end interval cover problem can be described as given a family $\mathbb{F}$ of paired-end intervals, find the least number of paired-end intervals of $\mathbb{F}$ to cover S. We prove that the paired-end interval cover problem is NP-complete. The c-interval cover problem is a generalization of paired-end interval cover that allows each member of the family $\mathbb{F}$ to have at most c disjoint intervals. It extends the classical set-cover problem reasonably. We show that the problem is APX-hard when $c \geq 3$. For solving these problems, we present a polynomial-time $6c$-approximation algorithm for the c-interval cover problem and a fixed parameter tractable algorithm for the k-bounded c-interval cover problem. Our implementation results show that the approximation ratio is much smaller than the theoretical bound for most real examples.

1 Introduction

The set-cover problem is a classical and fundamental problem in computer science with many applications. Given a finite set X of size n and a family $\mathbb{F}$ of subsets of X such that every element of X belongs to at least one subset in $\mathbb{F}$, the problem is to find a minimum-size subset $\mathbb{C} \subset \mathbb{F}$ whose members cover X. It is a well-known NP-complete problem showed in Karp's 21 NP-complete problems in 1972 and its solutions give rise to the development of the entire field of approximation algorithms [1]. The well known approximation algorithm for the set-cover problem which has a ratio of $1 + \ln n$ was given by Johnson [2] in 1974. Chvatal [3] improved the upper bound on the approximation ratio to $\ln n - \ln \ln n + O(1)$ in 1979. These ratios are tight because of a famous inapproximability result of Feige [4] which states that there is no $(1 - \varepsilon) \ln n$-approximation algorithm for the set-cover problem unless there are subexponential $O(n^{polylog(n)})$ time deterministic algorithms for problems in NP. A lower bound of $c \ln n$ was established by Raz and Safra [6] in 1997, where c is a constant, under the weaker assumption that P≠NP. A similar result with a higher value of c was recently proved by Alon, Moshkovitz and Safra [7] in 2006.

The general set-cover problem has two interesting variants: k-set cover and k-bounded set cover. We derive some results from these variants of set cover. Before presenting these results, we first give the formal definitions about the c-interval cover problem and the paired-end interval cover problem. Let $|A|$ be the size of set A, we have:

Definition 1. *Assume that S is a sequence that has no repetition with its elements, i.e. S is a permutation. Let c be an integer parameter.*

- *A* c-interval *$I = \{J_1, J_2, \cdots, J_t\}$ of S is a series of disjoint intervals $J_i (1 \leq i \leq t)$ of S which satisfies that $|I| = t \leq c$.*
- *The* c-interval cover problem *is defined as given a family $\mathbb{F}$ of c-intervals, find the least number of c-intervals of $\mathbb{F}$ to cover S.*

The paired-end interval cover problem is a special case of the c-interval cover problem for $c = 2$. For simplicity, we call the paired-end interval cover problem as PE-interval cover.

k-set cover restricts that every subset of X has size at most k. When $k = 2$, the 2-set cover can be reduced to maximum matching problem using the following two steps:

1. Find a maximum matching in a graph constructed according to the given 2-sets: create a vertex for each element, and there is an edge between two vertices if there is a 2-set consisting of this pair of elements.
2. Return all the 2-sets corresponding to the edges of the maximum matching and the 1-sets of the uncovered elements (by the collection of 2-sets which we found).

It is known that the maximum matching problem is solvable in polynomial time. So for PE-interval cover, if each interval is degenerated to a single point, it is equivalent to the 2-set cover and can be solved in polynomial time. However, we prove that the general PE-interval cover problem is NP-complete. Also, it is known that the k-set cover problem is APX-hard for $k \geq 3$ [22], the best known inapproximability bound for large k is $\Omega(\frac{k}{\ln k})$ [26]. We prove that the c-interval cover problem is APX-hard if $c \geq 3$. Moreover, a polynomial time $6c$-approximation algorithm is provided to solve the c-interval cover problem.

We define that one element of X occurs once if it appears in one of the subsets of $\mathbb{F}$. Then *k-bounded set cover* satisfies that the number of occurrences of any element of X in $\mathbb{F}$ is bounded by a constant $k \geq 2$ and there exists an element which appears in $\mathbb{F}$ exactly k times. The best known approximation algorithm which achieves the ratio $\frac{k}{k-(k-1)\sqrt[k]{1-\varepsilon}} + o(1)$ was presented by Reuven B.Y. and Zehavit K. [5] in 2004. In this paper, we show a fixed parameter tractable algorithm for k-bounded set cover problem, and hence it can be used directly to solve k-bounded c-interval cover.

Originally, the PE-interval cover problem is derived from pairwise end sequencing method which was developed based on shotgun sequencing. Shotgun sequencing is a method used for sequencing long DNA strands. Pairwise end sequencing method comes after shotgun sequencing, it is also known as double-barrel shotgun sequencing and was first described by Edwards and Caskey in 1991 [8]. In pairwise end sequencing, sequences are determined from both ends of random subclones derived from a DNA target. The benefit of this method toward shotgun sequencing is the information obtained by sequencing both ends of a fragment of DNA could be more useful. After the work of Edwards and Caskey, many variants of the strategy have been developed by several groups [9][10][11][12][13][14][15]. Pairwise end sequencing has been used successfully to sequence small genomic targets, such as microbial genomes and large-insert subclones of large genomes [16][17][18]. In 1999, heuristic algorithms were provided by Anson and Myers to handle whole genome shotgun sequencing [19]. In our paper, we generalize the PE-interval cover problem to the c-interval cover problem which extends the classical set-cover problem.

The rest of the paper are organized as follows: section 2 gives an approximation algorithm for the c-interval cover problem using the greedy method. The complexities of the PE-interval cover problem and the c-interval cover problem are presented in section 3. Then, some experimental results are presented in section 4. Section 5 concludes the paper.

2 A Greedy Approximation Algorithm for the *c*-Interval Cover Problem

We derive a $6c$-approximation algorithm for the c-interval cover problem using a greedy method. Theoretically, its approximation ratio is $6c$, but it has a much better approximation ratio in practice. We will show our experimental results in section 4.

Definition 2. *Let U be the set which includes all the uncovered elements of S, and let $I = \{J_1, J_2, \cdots, J_t\}$ be a c-interval of $\mathbb{F}$.*

- *An* improvement *of interval J_i for S is defined as $|J_i \cap U|$.*
- *A* single improvement *of c-interval I for S is defined as $\max_{1 \le i \le t} |J_i \cap U|$.*
- *A* sum improvement *of c-interval I for S is defined as $\sum_{1 \le i \le t} |J_i \cap U|$.*

The kernel of our greedy algorithm is that for each step, selecting a c-interval I_i in $\mathbb{F}$ such that I_i makes the biggest single improvement for S. The concrete steps are showed as follows:

Single Improvement Greedy Algorithm
Input: a genome sequence S and a family $\mathbb{F}$ of c-intervals $I_i (1 \le i \le |\mathbb{F}|)$.
Output: $6c$-approximation result
Steps:

```
1    Let $U \leftarrow S$, $C \leftarrow \emptyset$.
```

```
2   While $U \neq \emptyset$,
3       Select a $c$-interval $I_i$ from $\mathbb{F}$ such that a single interval $J_m$ of $I_i$
4           covers the most uncovered elements in $S$, which means $|J_m \cap U|$
5           is maximum, namely, the selection makes the biggest
6           single improvement for $S$.
7       $U \leftarrow U - \{\cup_{J_i \in I_i} J_i\}$
8       $C \leftarrow C \cup \{I_i\}$
9   Return $C$.
```
End of Algorithm

Since the number of iterations of the loop on lines 2-8 is bounded from above by $|F|$, and the loop body can be implemented to run in time $O(\sum_{I_i \in \mathbb{F}} |I_i|)$, there is an implementation that runs in time $O(|F| \sum_{I_i \in \mathbb{F}} |I_i|)$. The following theorem shows that it is a constant factor approximation algorithm.

Theorem 1. *There is a polynomial time* $6c$*-approximation algorithm for the* c*-interval cover problem.*

Proof. Let $\mathbb{O} = \{I_{O_1}, I_{O_2}, \cdots, I_{O_p}\}$ be an optimal solution for the c-interval problem, then $\mathbb{O}$ is made up of p c-intervals. Similarly, let $\mathbb{A} = \{I_{A_1}, I_{A_2}, \cdots, I_{A_q}\}$ be the approximation solution resulted from our greedy algorithm. Clearly, we have $q \geq p$. Consider a single interval J_{O_i} from a c-interval I_{O_i} in $\mathbb{O}$, J_{O_i} has two endpoints e_1 and e_2. For one of its endpoints e_1, since it is covered by the optimal solution $\mathbb{O}$, it must be covered by our approximation solution $\mathbb{A}$.

Let $\mathbb{A}_k$ be the partial approximation solution which is generated after k greedy selections of the algorithm and $U_k \subset S$ contains the elements which are not covered by the partial solution $\mathbb{A}_k$. Also let c-interval I_{k+1} be the $(k+1)$-th greedy selection which is added to the previous partial approximation solution $\mathbb{A}_k$ by our algorithm. Here we make k and I_{k+1} satisfy that after k selections, the next selection I_{k+1} is the first c-interval which covers e_1. We use $J_{e_1} \in I_{k+1}$ to represent the interval which covers e_1. There are two cases: the first case is that J_{e_1} is not the interval of I_{k+1} that covers the most number of uncovered elements of S, which means that $|J_{e_1} \cap U_k| \neq \max_{J_i \in I_{k+1}} |J_i \cap U_k|$; the second case is that J_{e_1} is the interval of I_{k+1} that covers the most number of uncovered elements of S, which means that $|J_{e_1} \cap U_k| = \max_{J_i \in I_{k+1}} |J_i \cap U_k|$.

Let $\mathbb{R} \subset \mathbb{F}$ contains all the remaining c-intervals which are still not selected by our greedy algorithm after $k+1$ selections, then we use $\mathbb{R}(e_1) \subset \mathbb{R}$ to represent the subset of $\mathbb{R}$ such that each c-interval of subset $\mathbb{R}(e_1)$ contains endpoints e_1. After the $(k+1)$-th selection of our algorithm, we count the number of greedy choices of c-intervals which contain the endpoint e_1. We define $c(e_1) = \{I_i : I_i$ is the k_0-th c-interval selected by the greedy algorithm, here $(k+1 \leq k_0 \leq |F|)$, and I_i contains an interval J_{k_0} with $e_1 \in J_{k_0}$ and $|J_{k_0} \cap U_{k_0-1}|$ maximal$\}$. According to the nature of greediness, we have:

Claim 1: $|c(e_1)| \leq 3$.

We explain the claim using two cases mentioned above:

– For case one, since $|J_{e_1} \cap U_k| \neq \max_{J_i \in I_{k+1}} |J_i \cap U_k|$, I_{k+1} cannot be counted into the number of greedy choices. Among all the c-intervals in set $\mathbb{R}(e_1)$, there is at most one greedy choice of c-interval $I_{a_1} \in c(e_1)$ such that I_{a_1} has non-zero improvements on both sides of interval J_{e_1}. There is at most one greedy choice of c-interval $I_{a_2} \in c(e_1)$ such that I_{a_2} has non-zero improvement on either side of interval J_{e_1}. Otherwise, it violates the greedy settings. Totally, our greedy approach has at most three choices which touch the endpoint e_1. So, $|c(e_1)| \leq 3$.
– For case two, since $|J_{e_1} \cap U_k| = \max_{J_i \in I_{k+1}} |J_i \cap U_k|$, we have $I_{k+1} \in c(e_1)$, which means that I_{k+1} should be counted into the number of greedy choices. Among all the c-intervals in set $\mathbb{R}(e_1)$, since c-interval I_{k+1} has already made the biggest single improvement for S, there is no other greedy choice of c-interval which belongs to $c(e_1)$ and has non-zero improvements on both sides of interval J_{e_1}. Similar to case one, there is at most one greedy choice of c-interval $I_{a_2} \in c(e_1)$ such that I_{a_2} has non-zero improvement on either side of interval J_{e_1}. Otherwise, it violates the greedy settings. Totally, our greedy approach has at most three choices which touch the endpoint e_1. So, $|c(e_1)| \leq 3$.

By summarizing the two cases, the claim 1 holds. Generally, for each endpoint e_i in some $I_{O_j} \in \mathbb{O}$, there are at most three greedy selections touching e_i. Namely, for some $J_{O_r} \in I_{O_j}$, there are at most three selections that contain one of its two endpoints. If we let $|\mathbb{A}|$ be the size of approximation solution, we have:

Claim 2: $|\mathbb{A}| \leq \sum_{e_i \in I_{O_j}, I_{O_j} \in \mathbb{O}} |c(e_i)|$ for $1 \leq i \leq 2$ and $1 \leq j \leq p$.

Claim 2 holds because any greedy selection of $\mathbb{A}$ must contain at least one endpoint of the optimal solution $\mathbb{O}$. Otherwise, if there is a greedy selection I_i in the approximation solution $\mathbb{A}$ which does not contain any endpoint of optimal solution $\mathbb{O}$, the greedy selection I_i must be a proper subset of an c-interval $I_{O_j} \in O$. However, because $I_i \subset I_{O_j}$, this selection violates the greedy settings and cannot be a correct greedy selection. Since for each endpoint e_i of $I_{O_j} \in \mathbb{O}$, $|c(e_i)|$ counts the number of greedy selections which contain the endpoint e_i. Thus by summing the number of greedy selections for each endpoint of $\mathbb{O}$, we have $|\mathbb{A}| \leq \sum_{e_i \in I_{O_j}, I_{O_j} \in \mathbb{O}} |c(e_i)|$ for $1 \leq i \leq 2$ and $1 \leq j \leq p$.

Since each $J_{O_r} \in I_{O_j}$ has two endpoints and each c-interval has at most c intervals, by Claim 1 and Claim 2, we have $|\mathbb{A}| \leq 2 \times 3 \times c \times |\mathbb{O}| = 6c|\mathbb{O}|$. $\square$

For the single improvement greedy algorithm, if we make a slight change, instead of selecting a c-interval that makes the biggest single improvement for S, we select a c-interval that makes the biggest sum improvement for S. This slight change makes the algorithm a special case of the greedy algorithm for the set-cover problem which has $1 + \ln n$ approximation ratio. If we call this algorithm *sum improvement greedy algorithm*, the same as set-cover, this algorithm has $1 + \ln |S|$ approximation ratio.

3 Complexity of the *c*-Interval Cover Problem

In this section, we explore the hardness of the c-interval cover problem. Theorem 3 shows that the problem is NP-complete when $c = 2$ and by Theorem 2, the c-interval cover problem is APX-hard if $c \geq 3$.

Definition 3. *[23] The* class APX *is the set of optimization problems that allow polynomial time approximation algorithms with approximation ratios bounded by a constant.*

– *If there is a polynomial time algorithm to solve a problem within any fixed percentage greater than zero, then the problem is said to have a* polynomial time approximation scheme (PTAS)*. Unless P=NP, there are problems that are in APX but not in PTAS; that is, problems that can be approximated within some constant factor, but not every constant factor.*
– *A problem is said to be* APX-hard *if there is a linear reduction from every problem in APX to that problem, and to be* APX-complete *if the problem is APX-hard and also in APX.*

It is well known that k-set cover problem is APX-hard for $k \geq 3$ [22], we have the following result:

Theorem 2. *The c-interval cover problem is APX-hard if* $c \geq 3$.

Proof. We have a simple polynomial time reduction from 3-set cover problem to 3-interval problem by setting each interval to be a single element. The same idea can be used to deduce solutions for $c > 3$. □

In the following part, we show the PE-interval cover problem is NP-complete. The core idea is that $(3, 3)$-SAT problem can be polynomial-time reducible to PE-Interval cover problem. We first give the definitions of 3SAT and $(3, 3)$-SAT.

Definition 4. *A* 3SAT instance *is a conjunctive form* $C_1 \cdot C_2 \cdots C_m$ *such that each* C_i *is a disjunction of at most three literals.* 3SAT *is the language of those* 3*SAT instances that have satisfiable assignments.*

Definition 5. *A* $(3, 3)$-SAT *instance is an instance G for* 3*SAT such that for each variable x, the total number of occurrences of x and $\bar{x}$ in G is at most* 3*, and the total number of occurrences of $\bar{x}$ in G is at most* 1*.* $(3, 3)$-SAT *is the language of those* $(3, 3)$*-SAT instances that have satisfiable assignments.*

For examples, $(\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1 \vee \bar{x}_3)$ is a 3SAT instance but not a $(3, 3)$-SAT instance since $\bar{x}_1$ appears twice in the formula. On the other hand, $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3)$ is both 3SAT and $(3, 3)$-SAT instance, and hence belongs to both 3SAT and $(3, 3)$-SAT. It is well known that 3SAT is a NP-complete problem, the following lemma shows that $(3, 3)$-SAT is polynomial-time reducible to 3SAT.

Lemma 1. *[20][21] There is a polynomial time reduction $f(.)$ from* 3*SAT to* $(3, 3)$*-SAT.*

Definition 6. *Given a* $(3,3)$*-SAT instance* $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$*, a clause* C_i *of* ϕ *is called* 3-positive clause *if it satisfies that* C_i *has three variables and all the three variables are positive. If a* $(3,3)$*-SAT instance* ϕ *has no 3-positive clause,* ϕ *is called a* $n3p$-$(3,3)$-SAT instance.

Lemma 2. *There is a polynomial time transformation from a* $(3,3)$*-SAT instance* ϕ *to a* $n3p$-$(3,3)$*-SAT instance* ϕ' *such that* ϕ *is satisfiable iff* ϕ' *is satisfiable.*

Proof. Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be an instance of $(3,3)$-SAT. The transformation from ϕ to ϕ' performs the same conversion for each clause $C_i (1 \leq i \leq m)$. If a clause $C_i = (x_1^i \vee x_2^i \vee x_3^i)$ of ϕ is a 3-positive clause, we convert C_i into $C_i' \wedge C_i'' = (x_1^i \vee x_2^i \vee \bar{x}_a^i) \wedge (x_3^i \vee x_a^i)$ by adding new variable x_a^i and letting $x_a^i = \bar{x}_3^i$. Since neither C' nor C'' ia a 3-positive clause, after the conversion, for each clause of ϕ, a $(3,3)$-SAT instance can be transformed into a $n3p$-$(3,3)$-SAT instance. It is easy to verify that C_i is true iff $C_i' \wedge C_i''$ is true, so ϕ is satisfiable iff ϕ' is satisfiable. The total transformation time is bounded by $O(m)$. □

For example, $\phi_1 = C_1 \wedge C_2 \wedge C_3$ is an instance of $(3,3)$-SAT, where $C_1 = (x_1 \vee \bar{x}_2 \vee x_3)$, $C_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$, $C_3 = (x_1 \vee x_2 \vee x_3)$. We convert it into a $n3p$-$(3,3)$-SAT instance $\phi_1' = C_1' \wedge C_2' \wedge C_3' \wedge C_4'$ by adding variable x_4 and letting $x_4 = \bar{x}_3$. Here $C_1' = (x_1 \vee \bar{x}_2 \vee x_3)$, $C_2' = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$, $C_3' = (x_1 \vee x_2 \vee \bar{x}_4)$, $C_4' = (x_3 \vee x_4)$.

As a decision problem, for a given permutation S, a family $\mathbb{F}$ of paired-end intervals and an integer number k, we ask simply whether there exists subset $\mathbb{C} \subseteq \mathbb{F}$ such that the permutation formed by subset $\mathbb{C}$ covers all the elements in S and the number of paired-end intervals in $\mathbb{C}$ is k.

Theorem 3. *The decision version of the PE-interval cover problem is NP-complete.*

Proof. To show that the PE-interval cover problem is in NP, for a given set $\mathbb{F}$ of paired-end intervals, we use subset $\mathbb{C} \subseteq \mathbb{F}$ as a certificate for $\mathbb{F}$. Consider S' as the set which contains all the elements that $\mathbb{C}$ covers. Checking the certificate $\mathbb{C}$ can be accomplished in polynomial time by checking whether, for each element $s \in S$, s belong to S'.

We next prove that $(3,3)$-SAT $\leq_p$ PE-interval cover, which shows that the PE-interval cover problem is NP-hard. Given a $(3,3)$-SAT instance $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m = s(x_1, x_2, \cdots, x_k)$ in which m is the number of clauses in ϕ and k is the number of variables in ϕ. Without loss of generality, we make three simplified assumptions about the formula ϕ. First, no clause contains both a variable and its negation, for such a clause, it is automatically satisfied by any assignment of values to the variables. Second, each variable appears in at least one clause, for otherwise it does not matter what value is assigned to the variable. Third, no variable appears three times in ϕ as positive, otherwise we can simply assign these variables true to remove them and the clauses which contains these variables without affecting the satisfiability of ϕ.

First, we need to transform ϕ into a $n3p$-$(3,3)$-SAT instance $\phi' = C'_1 \wedge C'_2 \wedge \cdots \wedge C'_{m'} = s(x'_1, x'_2, \cdots, x'_{k'})$ using Lemma 2. Let m' and k' be the clause size and variable size of ϕ' respectively, we have $m \leq m' \leq 2m$ $k \leq k' \leq k + m$. We construct a set $\mathbb{F}$ which contains $2k'$ paired-end intervals, ϕ is satisfiable iff there exists subset $\mathbb{C} \subseteq \mathbb{F}$ such that the sequence formed by subset $\mathbb{C}$ covers all the elements of S and the number of paired-end intervals in $\mathbb{C}$ is k'.

Then set $\mathbb{F}$ can be constructed using the following 4 steps of procedures (see an example showed in Figure 1):

Step1: Divide the sequence S into $m' + k'$ intervals $S_1, S_2, \cdots, S_{m'+k'}$ of same size $\frac{n}{m'+k'}$, we have $S_1 = \{1, 2, \cdots, \frac{n}{m'+k'}\}$ and $S_i = \{\frac{n(i-1)}{m'+k'}, \cdots, \frac{ni}{m'+k'}\}$ for $2 \leq i \leq m' + k'$.

Step2: Mark all the intervals and all the variables as unused, let $i = 1$.

Step3: For clause $C'_i = (l^i_1 \vee l^i_2 \vee l^i_3)$ of ϕ', here $i \leq m'$, we have three cases:

– *Case 1*: If C'_i has exactly one unused positive variable, select the first unused interval S_u. Let variables l^i_1, l^i_2 and l^i_3 represent the same interval S_u, then interval S_u is marked as used. We call the intervals such as S_u which are marked by clauses as *Clause Intervals*. For that unused positive variable $l^i_j (j \in \{1, 2, 3\})$ in C'_i, which means $l^i_j = x'_t (1 \leq t \leq k')$, select the first unused interval S_v, let l^i_j and $\bar{l}^i_j$ represent the same interval S_v. Then the variable $l^i_j = x'_t$ and the interval S_v are marked as used. We call the intervals such as S_v which are marked by variables as *Variable Intervals*.

– *Case 2*: If C'_i has two unused positive variables, select the second unused interval S_u, let variables l^i_1, l^i_2 and l^i_3 represent the same interval S_u. Then the interval S_u is marked as used. For that two positive variables $l^i_{j_1} (j_1 \in \{1, 2, 3\})$ and $l^i_{j_2} (j_2 \in \{1, 2, 3\})$ in C'_i, select the first and the second unused intervals S_{v_1}, S_{v_2}. Let $l^i_{j_1}$ and $\bar{l}^i_{j_1}$ represent the same interval S_{v_1}. Similarly, let $l^i_{j_2}$ and $\bar{l}^i_{j_2}$ represent the same interval S_{v_2}. Then the variables $l^i_{j_1}$, $l^i_{j_2}$ and the intervals S_{v_1}, S_{v_2} are marked as used.

– *Case 3*: If there dose not exist unused positive variable in C'_i, select the first unused interval S_u, let variables l^i_1, l^i_2 and l^i_3 represent the same interval S_u. Then the interval S_u is marked as used.

Let $i = i + 1$, repeat step 3.

Step4: If all the variables are marked as used, the construction is done. Otherwise there are unused variables, for each unused variable x_u, select the first unused interval S_u. Let x_u and $\bar{x}_u$ represent the same interval S_u.

It is easy to see that for each positive variable $x'_t (1 \leq t \leq k')$ in ϕ', set $\mathbb{F}$ has a paired-end interval, and for each negative variable $\bar{x}'_t (1 \leq t \leq k')$ in ϕ', set $\mathbb{F}$ also has a paired-end interval. So the total number of paired-end intervals in $\mathbb{F}$ is $2k'$. In addition, the cost of the reduction is bounded by $O(m' + k')$.

Finally, we must show that the transformation is a reduction. First, suppose that ϕ has a satisfying assignment. By Lemma 2, ϕ' also has a satisfying assignment. Then for each variable $x'_t (1 \leq t \leq k')$, x'_t is assigned either 1 or 0, if it is

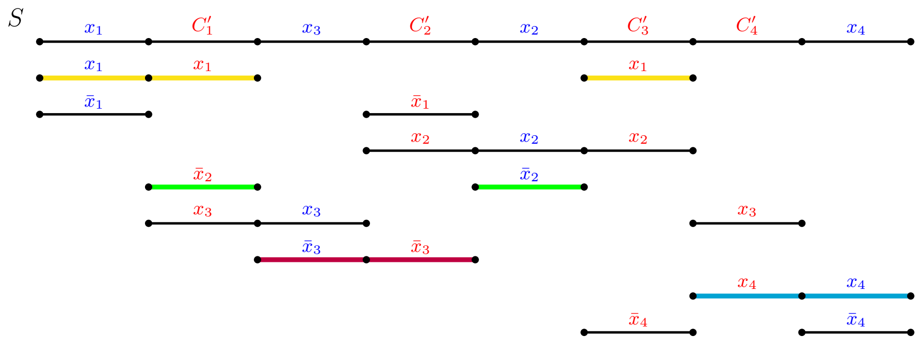


Fig. 1. The reduction from $n3p$-(3, 3)-SAT instance $\phi'_1 = C'_1 \wedge C'_2 \wedge C'_3 \wedge C'_4$ to an instance of PE-interval cover problem, where $C'_1 = (x_1 \vee \bar{x}_2 \vee x_3)$, $C'_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$, $C'_3 = (x_1 \vee x_2 \vee \bar{x}_4)$, $C'_4 = (x_3 \vee x_4)$. Set S is divided into eight intervals. Clause Intervals and Variable Intervals are marked as red and blue respectively. A satisfying assignment of the formula has $x_1 = 1$, $x_3 = 0$, $x_4 = \bar{x}_3 = 1$, and x_2 may be either 0 or 1. This assignment satisfies C'_1 and C'_3 with x_1, it satisfies C'_4 with x_4 and satisfies C_2 with $\bar{x}_3$, corresponding to four paired-end intervals with four different colored bold lines.

assigned 1, we select the paired-end interval represented by x'_t; otherwise if it is assigned 0, we select the paired-end interval represented by $\bar{x}'_t$. We put all the selected intervals into the set $\mathbb{C}$, it is obvious that $\mathbb{C}$ has k' elements. We need to show that the union of all the intervals in $\mathbb{C}$ covers the sequence S. Since ϕ' is satisfiable, each clause C'_i contains at least one literal l^i_j which is assigned 1. So for each clause interval, it is covered. For each variable interval, because either x'_t or $\bar{x}'_t$ is selected, it is covered too. The sequence S is made up of the union of all the clause intervals and variable intervals, therefore S is covered by $\mathbb{C}$.

Conversely, suppose that $\mathbb{F}$ is constructed using above procedures, $\mathbb{C} \subseteq \mathbb{F}$ is of size k' and it covers all the elements in S. We can assign 1 to all literals which are used to represent the k' paired-end intervals in $\mathbb{C}$. Here we do not need to worry about assigning 1 to both a literal and its complement. Since $\mathbb{C}$ is of size k', if the paired-end intervals represented by x'_t and $\bar{x}'_t$ are both included in $\mathbb{C}$, there must exist one variable x''_t, neither paired-end interval represented by x''_t or $\bar{x}''_t$ is included in $\mathbb{C}$. According to the construction of $\mathbb{F}$, the union of all intervals in $\mathbb{C}$ cannot cover all the elements of S. After assignment, each clause is satisfied. Then ϕ' is satisfied, therefore ϕ is satisfied. $\square$

We comment that this hardness result also holds if the reconstructed object is a sequence instead of a permutation.

4 Experimental Results

We implement the greedy algorithms described in Section 2 for the PE-interval cover problem. The experimental results of the single improvement greedy algorithm show a much better approximation ratio than $6c = 12$, which was given in Theorem 1. The sum improvement greedy algorithm is also tested. The experimental results show that it has a general better performance compared with the first one.

4.1 Implementation Details

With the purpose of easily testing the approximation performance, we select two short sequences of length $3,000$ and $6,000$. The selections are based on genome sizes of first sequenced RNA-genome[24] and DNA-genome[25]. In order to perform the test carefully, we create two parameters. The first one is *diSize* which represents the number of given 2-intervals, the second one is *maxInterLength*, it restricts the maximum length of each interval. The testing range for *diSize* is from 30 to $50,000$, while the testing range for *maxInterLength* is from 20 to sequence length. We first construct an optimal solution which uses the least number of 2-intervals to cover the whole sequence and add it into 2-interval set, and then the left 2-intervals are constructed randomly.

4.2 Results

For the single improvement greedy algorithm, the worst approximation ratio is 3.333. It is much better than the ratio 12 which was presented in Theorem 1. For the sum improvement greedy algorithm, the worst approximation ratio is 3.000. Comparing two greedy algorithms, their worst approximation ratios are almost the same. However, from their average approximation ratios, we discover that the second algorithm has an overall better performance compared with the first one. Though we cannot prove that the second algorithm is a constant factor approximation, the testing results show that it works very effectively in practice.

5 Conclusion

We explore the hardness of the PE-interval cover problem and present some approximation algorithms to solve it. There are still some unsolved problems. The sum improvement greedy algorithm has a better performance in our implementation; however, there is no approximation analysis of it. The approximation ratio given in Section 2 seems possible to be improvable. And we still do not know whether the PE-interval cover problem is APX-hard or not.

Acknowledgments. Bin Fu is supported in part by National Science Foundation Early Career Award 0845376. Binhai Zhu is partially supported by NSF of China under project 60928006.

References

1. Alon, N., Moshkovitz, D., Safra, S.: Algorithmic construction of sets for k-restrictions. ACM Trans. Algorithms (ACM) 2(2), 153–177 (2006)
2. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. System Sci. 9, 256–278 (1974)
3. Chvatal, V.: A greedy hueristic for the set-covering problem. Mathematics of Operations Research 4, 233–235 (1979)
4. Feige, U.: A threshold of ln n for approximating set cover. Journal of the ACM 45(4), 634–652 (1998)
5. Bar-Yehuda, R., Kehat, Z.: Approximating the dense set-cover problem. Journal of Computer and System Sciences 69, 547–561 (2004)
6. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: STOC 1997: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pp. 475–484. ACM, New York (1997)
7. Alon, N., Moshkovitz, D., Safra, S.: Algorithmic construction of sets for k-restrictions. ACM Trans. Algorithms 2(2), 153–177 (2006)
8. Edwards, A., Caskey, T.: Closure strategies for random DNA sequencing. Methods Comp. Methods Enzymol 3(1), 41–47 (1991)
9. Chen, E.Y., Schlessinger, D., Kere, J.: Ordered shotgun sequencing, as strategy for integrating mapping and sequencing of YAC clones. Genomics 17, 651–656
10. Richards, S., Muzny, D.M., Civitello, D.M., Lu, F., Gibbs, R.A.: Sequence map gaps and directed reverse sequencing for the completion of large sequencing projects. In: Automated DNA sequencing and Analysis, pp. 191–198. Academic Press, New York (1994)
11. Smith, M.W., Holmsen, A., Wei, Y.H., Peterson, M., Evans, G.A.: Genomic sequencing sampling: A strategy for high resolution sequence-based physical mapping of complex genomes. Nat. Genet. 7, 40–47 (1994)
12. Venter, J.C., Adams, M.D., Sutton, G.G., Kerlavage, A.R., Smith, H.O., Hunkapiller, M.: Shotgun sequencing for the human genome. Science 280, 1540–1542 (1998)
13. Venter, J.C., Smith, H.O., Hood, L.: A new strategy for genome sequencing. Nature 381, 364–366 (1996)
14. Weber, J.L., Myers, E.W.: Human whole-genome shotgun sequencing. Genome Res. 7, 401–409 (1997)
15. Siegel, A.F., Engh, G., van den Hood, L., Trask, B., Roach, J.C.: Analysis of sequence-tagged connector (STC) strategies for DNA sequencing. Genomics 68(3), 237–246 (2000)
16. Eward, A., Voss, H., Rice, P., Civitello, A., Stegemann, J., Schwager, C., Zimmerman, J., Erfle, H., Caskey, T., Ansorge, W.: Automated DNA sequecing of the human HPRT locus. Genomics 6, 593–608 (1990)
17. Fleischmann, R.D., Adams, M.D., White, O., et al.: Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. Science 269(5223), 496–498, 507-512 (1995)
18. Fraser, C.M., Gocayne, J.D., White, O., et al.: The minimal gene complement of Mycoplasma genitalium. Science 270(5235), 397–404 (1995)
19. Anson, E., Myers, G.: Algorithms for whole genome shotgun sequencing. In: Proceedings of the Third Annual International Conference on Computational Molecular Biology (1999)

20. Tovery, C.A.: A simplified satisfiability problem. Discrete Applied Mathematics 8(85), 89 (1984)
21. Fu, B.: Multivariate Polynomial Integration and Derivative Are Polynomial Time Inapproximable unless P=NP. Electronic Colloquium on Computational Complexity (ECCC) 17, 202 (2010)
22. Kann, V.: Maximum bounded 3-dimensional matching is MAX SNP-complete. Information Processing Letters 37, 27–35 (1991)
23. Papadimitriou, C., Yannakakis, M.: Optimization, approximation and complexity classes. Journal of Computer and System Sciences 43, 425–440 (1991)
24. Fiers, W., et al.: Complete nucleotide-sequence of bacteriophage MS2-RNA - primary and secondary structure of replicase gene. Nature 260(5551), 500–507 (1976)
25. Sanger, F., Air, G.M., Barrell, B.G., Brown, N.L., Coulson, A.R., Fiddes, C.A., Hutchison, C.A., Slocombe, P.M., Smith, M.: Nucleotide sequence of bacteriophage phi X174 DNA. Nature 265(5596), 687–695 (1977)
26. Hazan, E., Safra, S., Schwartz, O.: On the complexity of approximating k-set packing. Computational Complexity 15(1), 20–39 (2006)

Exponential and Polynomial Time Algorithms for the Minimum Common String Partition Problem

Bin Fu[1], Haitao Jiang[2], Boting Yang[3], and Binhai Zhu[4]

[1] Department of Computer Science, University of Texas-Pan American
Edinburg, TX 78539, USA
binfu@cs.panam.edu

[2] School of Computer Science and Technology, Shandong University,
Jinan Shandong, 250100, China
htjiang@mail.sdu.edu.cn

[3] Department of Computer Science University of Regina Regina
Saskatchewan Canada, S4S 0A2, Canada
boting@cs.uregina.ca

[4] Department of Computer Science, Montana State University
Bozeman, MT, 59717-3880, USA
bhz@cs.montana.edu

Abstract. Given two strings S and S' of the same length, the Minimum Common String Partition (MCSP) is to partition them into the minimum number of strings $S = S_1 \cdot S_2 \cdots S_k$ and $S' = S'_1 \cdot S'_2 \cdots S'_k$ such that the substrings $\langle S'_1, S'_2, \cdots, S'_k\rangle$ is a permutation of $\langle S_1, S_2, \cdots, S_k\rangle$. MCSP is an NP-complete problem originating from computational genomics. There exists constant-factor approximations for some special cases, but the factors are impractical. On exact solutions, it is open whether there exists an FPT algorithm for the general case and some inefficient FPT algorithms for very special cases. In this paper, we present an $O(2^n n^{O(1)})$ time algorithm for the general case. We also show an $O(n(\log n)^2)$ time algorithm which solves the case for almost all strings S and S' if the length of each block in their minimum common partition is at least $\frac{d_0 \log n}{\log t}$ for some positive constant d_0, where t is the size of the alphabet Σ.

1 Introduction

The minimum common string partition (MCSP) problem has attracted a lot of attention, partly because of its applications in computational biology, text processing and data compression. More specifically, MCSP has a close connection with the genome rearrangement problems such as edit distance and sorting by reversals, etc. In fact, MCSP was initially studied exactly as a problem on "edit distance with moves" [3,15].

A partition P of a string X is a string $P = \langle P_1, P_2, \ldots, P_m\rangle$ of strings whose concatenation is equal to X, that is $P_1P_2 \ldots P_m = X$. Each of the strings P_i is called a block of P. Given a partition P of a string X and a partition Q of a string Y, we say that the pair $\pi = (P, Q)$ is a common partition of X and Y if Q is a permutation of P, that is, there exists a permutation σ on $[m]$ such that $P_i = Q_{\sigma_i}, 1 \leq i \leq m$. The *minimum common string partition problem* is to compute a common partition of X, Y with the minimum number of blocks.

Naturally, in the Minimum Common String Partition (MCSP) problem, we are given two strings X and Y of length n over an alphabet Σ. Let each symbol appear the same number of times in X and Y. Throughout this paper, we assume that X and Y always satisfy this condition. This is certainly a sufficient and necessary condition for us to compute a common string partition for X and Y. For example, two strings $X = beabcdb$ and $Y = abdbebc$ have a common partition $(\langle b, e, ab, c, db\rangle, \langle ab, db, e, b, c\rangle)$. There are several variants of MCSP. We call the restricted version where each letter occurs at most d times in each input string as d-MCSP. When the input strings are over an alphabet of size c, we call the corresponding problem $MCSP^c$.

The problem d-MCSP has been well studied. 2-MCSP (and therefore MCSP) is NP-hard; moreover, APX-hard [9]. Several approximation algorithms have been proposed for MCSP problem [9,12]. Chen et al. [1] studied the problem of computing signed reversal distance with duplicates (SRDD). They introduced the signed minimum common partition problem as a tool for dealing with SRDD and observed that for any two related signed strings X and Y, the size of a minimum common partition and the minimum number of reversal operations needed to transform X and Y, are within a factor of 2. Kolman and Walen [14] devised an $O(d^2)$-approximation algorithm running in $O(n)$ time for SRDD.

Chrobak et al. [2] analyzed the greedy algorithm for MCSP, they showed that for 2-MCSP the approximation ratio is exactly 3, for 4-MCSP the approximation ratio is $\Omega(\log n)$; for the general MCSP, the approximation ratio is between $\Omega(n^{0.43})$ and $O(n^{0.67})$. Kaplan and Shafrir [11] improved the lower bound to $\Omega(n^{0.46})$ when the input strings are over an alphabet of size $O(\log n)$. Kolman [13] described a simple modification of the greedy algorithm; the approximation ratio of the modified algorithm is $O(p^2)$ for p-MCSP. Most recently, Goldstein and Lewenstein presented an greedy algorithm for MCSP [8].

In the framework of parameterized complexity, Damaschke first solved MCSP by an FPT algorithm with respect to parameters k (size of the optimum solution), r (the repetition number) and t (the distance ratio, depending on the shortest block in the optimum solution) [4]. Jiang et al. showed that both d-MCSP and $MCSP^c$ admit FPT algorithms when d and c are (constant) parameters [10]. However, it remains open whether MCSP admits any FPT algorithm parameterized only on k.

In this paper, we show an $O(2^n n^{O(1)})$ time algorithm to give an exact solution for the general problem. We also develop an $O(n(\log n)^2)$ time algorithm for almost all strings S and S' if the length of each block in their minimum common partition is at least $\frac{d_0 \log n}{\log t}$ for some positive constant d_0, where t is the size of the alphabet Σ.

The paper is organized as follows: In section 2, we show an $O(2^n n^{O(1)})$ time algorithm to solve the minimum common string partition problem. In section 3, we show a polynomial time algorithm to solve the minimum common string partition problem for almost all strings.

2 An $O(2^n n^{O(1)})$ Time Algorithm for General Cases

In this section, we give an exact solution for the minimum common string partition problem. Our algorithm runs in $O(2^n n^{O(1)})$ time. Our method is to convert it into a detection of a multilinear monomial in the sum of product expansion of a multivariate polynomial.

Definition 1. *Let $S = a_1 a_2 \cdots a_n$ be a string of length n.*

- *For an integer $i \in [1, n]$, define $S[i] = a_i$.*
- *For an interval $[i, j] \subseteq [1, n]$, define $S[i, j]$ to be the substring $a_i a_{i+1} \cdots a_j$ of S.*

Definition 2. *Given a pair of strings S and S', the common string partition problem is to find partitions for $S = S_1 \cdots S_k$ and $S' = S'_1 \cdots S'_k$ such that $S'_1, \cdots, S'_k$ is a rearrangement of $S_1, \cdots, S_k$, and k is the least.*

We will use a multilinear monomial to encode a solution for the partition problem. We first define the monomial and multilinear monomial in a multivariate polynomial as follows.

Definition 3. *Assume that $x_1, \cdots, x_k$ are variables.*

- *A monomial has format $x_1^{a_1} x_2^{a_2} \cdots x_k^{a_k}$, where a_i are nonnegative integers.*
- *A multilinear monomial is a monomial such that each variable has degree at most one. For examples, $x_3 x_5 x_6$ is a multilinear monomial, but $x_3 x_5^3 x_6^2$ is not.*
- *For a polynomial $p(x_1, \cdots, x_k)$, its sum of product expansion is $p(x_1, \cdots, x_k) = \sum_j c_j q_j(x_1, \cdots, x_k)$, where each $q_j(x_1, \cdots, x_k)$ is a monomial with c_j as its coefficient.*

Theorem 1. *There is an $O(2^n n^{O(1)})$ time algorithm for the minimum common string partition problem.*

Proof. Let S and S' be two input strings of length n. Let each position i of S' have a unique variable x_i to represent it. Define $P[a, b] = x_a x_{a+1} \cdots x_b$. For each interval $[s, t]$, define $G_{s,t} = \sum_{S'[s',t'] = S[s,t]} P[s', t']$.

For each $i \le n$, define a polynomial $F_{i,1} = G_{1,i}$. It represents all the substrings of S' that match $S[1, i]$. This is formally stated by Claim 1 below.

Claim 1. $S'[u, v]$ is a substring of S' with $S'[u, v] = S[1, i]$ if and only if there is a multilinear monomial $P[u, v]$ in $F_{i,1}$.

Let $F_{0,t} = 1$ for each $t \ge 1$. For each $i \le n$, define

$$F_{i,t+1} = \sum_{1 \le j \le i} G_{j,i} \cdot F_{j-1,t}. \tag{1}$$

This polynomial has the property for encoding some partial common partition. It is stated in Claim 2.

Claim 2. Let i be an arbitrary integer parameter in the range $[1, n]$. Then there is a partition $S[i_1, j_1], \cdots, S[i_t, j_t]$ for $S[1, i]$ such that $S[i_r, j_r] = S'[i'_r, j'_r]$ for some disjoint intervals $[i'_1, j'_1], \cdots, [i'_t, j'_t]$ that are subset of $[1, n]$ if and only if there is a multilinear monomial Q in the sum of product expansion of $F_{i,t}$ with $Q = \prod_{r=1}^{t} P[i'_r, j'_r]$.

Proof. We prove this claim by induction. For the case $t = 1$, it follows from Claim 1. Assume that the claim is true for t.

We consider the case $t + 1$. We have the following two cases for the induction.

– Assume that there is a partition $S[i_1, j_1], \cdots, S[i_{t+1}, j_{t+1}]$ for $S[1, i]$ such that $1 = i_1 \leq j_1 = i_2 - 1 < j_2 = \cdots j_t = i_{t+1} - 1 < j_{t+1} = i$, and $S[i_r, j_r] = S'[i'_r, j'_r]$ for $r = 1, \cdots, t + 1$ with some disjoint intervals $[i'_1, j'_1], \cdots, [i'_{t+1}, j'_{t+1}]$. By our inductive hypothesis, there is a multilinear monomial $Q' = \prod_{r=1}^{t} P[i'_r, j'_r]$ that is in the sum of product expansion of $F_{j_t,t}$. By the definition of $G_{i_{t+1},j_{t+1}}$, $G_{i_{t+1},j_{t+1}}$ contains the multilinear monomial $P[i'_{t+1}, j'_{t+1}]$ in its sum of product expansion. By the definition of $F_{i,t+1}$ in equation (1), we have multilinear monomial $Q = Q' \cdot P[i'_{t+1}, j'_{t+1}]$ in the sum of expansion of $F_{i,t+1}$.
– Assume that there is multilinear monomial Q in $F_{i,t+1}$. By the definition of $F_{i,t+1}$ in equation (1), there is an integer j and an interval $[i'_{t+1}, j'_{t+1}]$ such that $S[j, i] = S'[i'_{t+1}, j'_{t+1}]$, and there is a multilinear monomial Q' with $Q = P[i'_{t+1}, j'_{t+1}] \cdot Q'$.
  By the inductive hypothesis, there is a partition $S[i_1, j_1], \cdots, S[i_t, j_t]$ for $S[1, j - 1]$ such that for each $1 \leq r \leq t$, $S[i_r, j_r] = S'[i'_r, j'_r]$ for some disjoint intervals $[i'_1, j'_1], \cdots, [i'_t, j'_t]$, and $Q' = \prod_{r=1}^{t} P[i'_r, j'_r]$. Since Q is a multilinear monomial, we know that there is no intersection between $[i'_{t+1}, j'_{t+1}]$ and $\cup_{r=1}^{t} [i'_r, j'_r]$. □

By Claim 2, it is easy to see that there is a partition of k segments if and only if there is a multilinear monomial $x_1 x_2 \cdots x_n$ in the sum of product expansion of $F_{n,k}$.

We have an $O(2^n n^{O(1)})$ time algorithm to find the multilinear monomial by evaluating $F_{n,k}$ from bottom-up. During the evaluation, we only keep the multilinear monomials in $F_{i,t}$. Since the total number of variables is n, the total number of multilinear monomials is at most 2^n. We note that $G_{j,i}$ has at most $O(n)$ monomials. Therefore, each multiplication $G_{j,i} F_{j-1,t}$ takes $O(2^n n^{O(1)})$ time. The arithmetic expression for $F_{n,k}$ involves $n^{O(1)}$ $+$ and $\cdot$ operations. Therefore, the total time for generating all the multilinear monomials in the sum of product expansion of $F_{n,k}$ is $O(2^n n^{O(1)})$.

Testing the existence of multilinear monomial $x_1 x_2 \cdots x_n$ in the sum of product expansion over $F_{n,k}$ gives the existence of a common partition of at most k blocks. Tracing how the multilinear monomial $x_1 x_2 \cdots x_n$ is formed with a bottom-up approach to find all multilinear monomial in the sum of product expansion of $F_{n,k}$ shows how a common partition of at most k blocks is computed. This can be seen in the proof of Claim 2. $\square$

3 Polynomial Time Algorithm for Almost All Cases

In this section, we present an $O(n(\log n)^2)$ time algorithm which solves the case for almost all strings S and S' if the length of each block in their minimum common partition is at least $d_0 \log n$ for some positive constant d_0.

Definition 4. *Assume that strings S and S' are two strings of the same length. A k-common partition divides S into k blocks $S_1, \cdots, B_k$, and S' into another k blocks $S'_1, \cdots, S'_k$ such that $S'_1, \cdots, S_k$ is a permutation of $S_1, \cdots, S_k$. A (k, m)-common partition S and S' is a k-common partition that each block is of length at least m.*

Definition 5. *Assume that strings S and S' are two strings of the same length, and there is a minimum common partition that divides S into $S_1, \cdots, S_k$, and S' into $S'_1, \cdots, S'_k$ such that $S_j = S'_{i_j}$ for $j = 1, \cdots, k$. Let $S_j = S[a_j, b_j]$ for $j = 1, \cdots, k$, and $S'_j = S'[a'_j, b'_j]$ for $j = 1, \cdots, k$.*

- *A (k, m)-rough common partition is to approximate the k pairs of matched right boundaries (b_j, b'_{i_j}) with $(end[j], end'[i_j])$ such that $end[j] \in (b_j - m, b_j + m)$ and $end'[i_j] \in (b'_{i_j} - m, b'_{i_j} + m)$ for $j = 1, 2, \cdots, k$.*
- *A boundary determination for a (k, m)-rough common partition of S and S' is that given a (k, m)- rough common partition of S and S' (with $(end[j], end'[i_j])$ for $j = 1, 2, \cdots, k$), derive a k-common partition $S_1, S_2, \ldots, S_k$ of S and $S'_1, S'_2, \ldots, S'_k$ of S' such that $S_i = S[x_i, y_i]$, $S'_i = S[x'_i, y'_i]$ with $y_i \in (end[i] - m, end[i] + m)$ and $y'_i \in (end'[i] - m, end'[i] + m)$for $i = 1, 2, \ldots, k$.*

Definition 6. *Let Σ be an alphabet with at least 2 characters. Let h and n be integers with $h \leq n$. Define $\Phi_{n,h}(\Sigma)$ be the set of all strings S in Σ^n such that $S[i, i+h-1] = S[j, j+h-1]$ for some two indices $i \neq j$ with $1 \leq i < j \leq n-h+1$. Define $\Psi_{n,h}(\Sigma) = \cup_{u=h}^{n} \Phi_{n,u}(\Sigma)$.*

Lemma 1. *Let $t = |\Sigma| \geq 2$. $\frac{|\Psi_{n,h}(\Sigma)|}{|\Sigma^n|} \leq n^2 t^{-(\lfloor \frac{h}{2} \rfloor - 1)}$.*

Proof. Assume that S is a random string of Σ^n. For two pairs $(S[i+r], S[j+r])$ and $(S[i+r'], S[j+r'])$ of characters in S, they are independent if $\{i+r, j+r\} \cap \{i+r', j+r'\} = \emptyset$. We can pick up $\lfloor \frac{u}{2} \rfloor$ independent pairs $(S[i+r_i], S[j+r_i])$, for $i = 1, \cdots, \lfloor u/2 \rfloor$. Let H be the set of those independent pairs. Set H contains at least $\frac{u}{2}$ independent pairs.

Let $u \geq h$. We consider $S[i, i+u-1]$ and $S[j, j+u-1]$. The probability that the k-th characters of these two substrings are the same is $\frac{1}{|\Sigma|} = \frac{1}{t}$. With probability at most $t^{-\lfloor \frac{u}{2} \rfloor}$, $(S[i+r] = S[j+r])$ for all independent pairs $(S[i+r], S[j+r])$ in H. With probability at most $t^{-\lfloor \frac{u}{2} \rfloor}$, $(S[i, i+u-1] = S[j, j+u-1])$. Therefore, with probability at most $n^2 t^{-\frac{u}{2}}$, there exist i and j with $1 \le i < j \le n-u+1$ such that $S[i, i+u-1] = S[j, j+u-1]$).

Therefore, with probability at most $\sum_{u=h}^{n} n^2 t^{-\lfloor \frac{u}{2} \rfloor} \le n^2 t^{-(\lfloor \frac{h}{2} \rfloor - 1)}$, there are integers i, j and u with $1 \le i < j \le n-u+1$ and $h \le u \le n$ such that $S[i, i+u-1] = S[j, j+u-1]$. □

For an intuitive understanding of Lemma 1, note that $\Psi_{n,h}$ is a subset of Σ^n. If y is constant, we can select a constant c_0 such that $n^2 t^{-(\lfloor \frac{h}{2} \rfloor - 1)} < \frac{1}{2^y}$ with $h \ge \frac{c_0 \log n}{\log t}$. Therefore, $\Psi_{n,h}(\Sigma)$ is a small subset of strings in Σ^n which only has $\frac{1}{2^y}$ fraction of strings in Σ^n when constant y is selected large enough. The algorithm and its performance is stated by Thereom 2. Its proof will be given later in this section.

Lemma 2. *If there is a (k, h)-common partition for S and S', then S is in $\Psi_{n,h}$ implies S' is in $\Psi_{n,h/4}$.*

Proof. Assume that there is a (k, h)-common partition $S = S_1 \cdots S_k$ and $S' = S'_1 \cdots S'_k$ such that $S'_1, \cdots, S'_k$ is a rearrangement of $S_1, \cdots, S_k$. Assume that there are two identical substrings $V_1 = S[i, j]$ and $V_2 = S[u, v]$ of length h in S. There is a block S_t that contains at least half of the substring V'_1 of V_1. Let V'_2 be the substring of V_2 equal to V'_1. There is a block S_x that contains at least half substring V''_2 of V'_2. Let V''_1 be the substring of V'_1 equal to V''_2. The length of V''_1 and V''_2 is at least $\frac{h}{4}$. Both V''_1 and V''_2 appear as substrings in S' since they are substrings of some blocks in S. □

Theorem 2. *There is an $O(nh^2)$ time algorithm such that given two input strings in $\Sigma^n - \Psi_{n,h}$ that a minimum common partition between them is a $(k, 3h)$-common partition, then it outputs the $(k, 3h)$-common partition between them.*

Before we present the formal description of our algorithm, we give a brief idea of the algorithm. Let $S = S_1 \cdots S_k$ and $S' = S'_1 \cdots S'_k$ be the $(k, 3h)$-common partition, which is also a minimum common string partition. Assume S_j matches S'_{i_j} for $j = 1, 2, \cdots, k$. Use the first substring $S_1[1, h]$ in a block of S_1 to match $S'_{i_1}[1, h]$ in block S'_{i_1}. The two matched regions $S_1[1, h]$ and $S'_{i_1}[1, h]$ are extended to $S_1[1, m_1]$ and $S_{i_1}[1, m_1]$ until the rough end of the two matched blocks S_1 and S'_{i_1} ($m_1 \in (|S_1| - h, |S_1|]$). With $m_1 \in (|S_1| - h, |S_1|]$, we have that $S[m_1 + h, m_1 + h + h - 1] = S_2[x, x + h - 1]$ for some $x \in [1, h]$. Substring $S_2[x, x + h - 1]$ is at the rough starting area of S_2 and is used to match the rough start area $S'_{i_2}[x, x + h - 1]$ of the block in S'_{i_2}. After $S'_{i_2}[x, x + h - 1]$ is found in S', extend them to $S_2[x, m_2]$ and $S'_2[x, m_2]$ with m_2 close to the length of S_2 ($m_2 \in (|S_2| - h, |S_2|]$). Repeat the above process until we obtain a rough match for the minimum common string partition. In the next phase, we detect the exact boundaries from the rough boundaries.

Lemma 3. *There is an $O(nh)$ time algorithm to determine if a string S is in $\Phi_{n,h}(\Sigma)$.*

Proof. Let S be a string of length n. Derive all substrings of length h of S: $S[1,h], S[2,h+1], \cdots, S[n-h+1,n]$. Sort them by the alphabetic order, and check if any two of them are the same. Two substrings are equal if and only if S a $\Phi_{n,h}(\Sigma)$. We just use the standard radix sorting which needs $O(nh)$ time. □

```
LocalMatch(i, j)
Input: $i$ is a position in $S$ and $j$ is a position in $S'$.
Steps:

(1)    Let $p = i$;
(2)    Let $q = j$;
(3)    Repeat
(4)        if $S[p, p+h-1] = S'[q, q+h-1]$
           then let $p = p+h$ and $q = q+h$;
(5)    Until $S[p, p+h-1] \neq S'[q, q+h-1]$;
(6)    Return $(p-h, q-h)$;

End of LocalMatch
```

We have a module that gives a rough common partition between two strings. The boundaries among those blocks will be determined by another module Boundary(.).

```
RoughBoundaries($S, S'$)
Input: two strings $S$ and $S'$.
Steps:

(1)     Derive $s_1 = S[1,h], \cdots, s_r = S[r, r+h-1]$ from $S$ with $n = r+h-1$;
(2)     Derive $s'_1 = S'[1,h], \cdots, s'_r = S'[r, r+h-1]$ from $S'$ with $n = r+h-1$;
(3)     Sort $s_1, \cdots, s_r, s'_1, \cdots, s'_r$
        Find the string $s'_j = S'[x,y]$ with $s_1 = s'_j$ via the sorted list;
(4)     Let $start[1] = 1$; (the starting point of the first block in $S$)
(5)     Let $i = 1$;
(6)     Let $z = 1$;
(7)     Let $w = x$;
(8)     Repeat
(9)         Let $(u,v) = LocalMatch(z,w)$;
(10)        Let $end[i] = u$; (the rough ending point of block $i$ in $S$)
(11)        Let $end'[i] = v$; (the rough ending point of block in $S'$ matching $i$-th block)
(12)        Let $start[i+1] = end[i] + h$; (the rough starting point of block $i+1$ in $S$)
(13)        Let $i = i+1$;
(14)        Let $z = u+h$;
(15)        Find the string $s'_j = S'[x,y]$ with $s'_j = s_z = S[z, z+h-1]$ from the sorted list;
```

```
(16)            Let $w = x$;
(17)        Until a $(k, h)$ rough common partition is reached between $S$ and
     $S'$ for some integer $k$;
     return the rough common partition .
```

End of RoughBoundaries

Lemma 4. *Assume that S and S' are in $\Sigma^n - \Psi_{n,h}$. The algorithm RoughBoundaries(S, S') gives a (k, h)-rough common partition in $O(nh^2)$ time if k is the least number of blocks for the minimum common string partition problem.*

Proof. Assume that the optimal solution has the partition $S = S_1^o \cdots S_k^o$ and $S' = S_1'^o \cdots S_k'^o$. We assume that $S_j = S'_{i_j}$ for $j = 1, \cdots, k$. Let $S_i^o = S[a_i, b_i]$ and $S_i'^o = S[a'_i, b'_i]$ for $i = 1, \cdots, k$.

We will show that the rough common partition derived by the algorithm is close to the boundaries in the optimal solution. We prove the following claim by induction.

Claim. For $j \le k$, $b_j - h < end[j] < b_j + h$ and $b'_{i_j} - h < end'[i_j] < b'_{i_j} + h$.

For $j = 1$, We start from the $S[1, h]$, which is equal to $S_1^o[1, h]$, and can match $S'_{i_1}[1, h]$. By calling LocalMatch(.), it returns (p, q) for two positions in S and S', respectively. We have that p is at least $b_j - h + 1$. Otherwise, p can be increased in the LocalMatch(.). On the other hand, we also have $p < a_2 + h - 1$. Otherwise, we have a substring S_a of length h in S_2 matching another substring S'_a of length h in S'. Thus, S_a must be in S'_{i_2} (otherwise, S' contains two S_as at two different locations, which contradicts that S' is in $\Sigma^n - \Psi_{n,h}$). Thus, the S_1 and S_2 can be merged into a single block, and bring a common partition with $k - 1$ blocks. This contradicts that k is the least number of blocks for the common partition. Thus, $p \in (b_1 - h, b_1 + h)$, and $q \in (b'_{i_1} - h, b'_{i_1} + h)$. Therefore, $end[1]$ is in $(b_1 - h, b_1 + h)$, and $end'[i_1]$ is in $(b'_{i_1} - h, b'_{i_1} + h)$. Thus, we have $start[2]$ is in $[a_2, a_2 + 2h)$ since $a_2 = b_1 + 1$.

Assume that $b_j - h \le end[j] \le b_j$ and $start[j + 1]$ is in $[a_{j+1}, a_{j+1} + 2h)$. By calling LocalMatch(.), it returns (p, q) for two positions in S and S', respectively. By a similar argument as the case at $j = 1$, we also have $end[j + 1]$ is in $(b_{j+1} - h, b_{j+1} + h)$ and $end'[i_{j+1}]$ is in $(b'_{i_{j+1}} - h, b'_{i_{j+1}} + h)$. If $j + 1 < k$, we have $start[j + 2]$ is in $[a_{j+2}, a_{j+2} + 2h)$ since $a_{j+2} = b_{j+1} + 1$. □

After determining the rough common partition , we need one more module to determine the boundaries.

Definition 7. *Assume that strings S and S' have minimum common partition $S = S_1 S_2 \ldots S_k$, and $S' = S'_1 S'_2 \ldots S'_k$. Each S_j in S matches S'_{i_j} in S'. For each block S_j, let $L(j)$ be the left boundary point of S_i in S, and $R(j)$ be the right boundary point for S_j in S. $L'(j)$ and $R'(j)$ are defined similarly for S'.*

– *For each boundary point p in one of the blocks in S and S', define $A(p)$ recursively:*
  - $p \in A(p)$.
  - *If $L(j) \in A(p)$, then $L'(i_j) \in A(p)$;*
  - *If $L'(i_j) \in A(p)$, then $L(j) \in A(p)$;*
  - *If $R(j) \in A(p)$, then $R'(i_j) \in A(p)$;*
  - *If $R'(i_j) \in A(p)$, then $R(j) \in A(p)$;*
  - *If $L(j) \in A(p)$ and $j > 1$, then $R(j-1) \in A(p)$;*
  - *If $R(j) \in A(p)$ and $j < k$, then $L(j+1) \in A(p)$;*
  - *If $L'(j) \in A(p)$ and $j > 1$, then $R'(j-1) \in A(p)$; and*
  - *If $R'(j) \in A(p)$ and $j < k$, then $L'(j+1) \in A(p)$.*

– *For a position i, and a boundary point p in S or S', let $A(p)[i]$ is to assign i to p, and derive the consequent positions for the points in $A(p)$.*

– *For a position i, and a boundary point p in S or S', let $A(p)[i]$ is m-consistent if the following conditions hold:*
  - *For each left boundary $q \in A(p)$ in some block in S, $S[q, q+m-1] = S'[q', q'+m-1]$, where q' is corresponding left boundary for a block in S';*
  - *For each left boundary $q' \in A(p)$ in some block in S', $S[q, q+m-1] = S'[q', q'+m-1]$, where q is corresponding left boundary for a block in S;*
  - *For each right boundary $q \in A(p)$ in some block in S, $S[q-m+1, q] = S'[q'-m+1, q']$, where q' is corresponding right boundary for a block in S'; and*
  - *For each right boundary $q' \in A(p)$ in some block in S', $S[q-m+1, q] = S'[q'-m+1, q']$, where q is corresponding right boundary for a block in S.*

```
Boundary($k, h, RB$)
Input: $RB$ is a $(k,h)$-rough common partition for $S = S_1, \cdots, S_k$ and $S' = S'_1, \cdots, S'_k$ with $S_j = S'_{i_j}$ for $j = 1, \cdots, k$. Let $start[j]$ ($end[j]$) be the rough left (right) boundary point of $S_j$ in $S$. Let $start'[j]$ ($end'[j]$) be the rough left (right) boundary point of $S'_j$ in $S'$.
Steps:
    Select $A(R(j_1)), \cdots, A(R(j_t))$ that are disjoint each other and $\cup_{r=1}^{t} A(R(j_r))$ includes all right boundaries of those $k$ blocks $S_1, \cdots, S_k$;
    For $r = 1$ to $t$
        Find $j$ in $(end[j_r] - h, end[j_r] + h)$ such that $A(R(j_r))[j]$ is $2h$-consistent;
        If $j$ is found
        then fix the boundary points of $A(R(i))$ according to the values in $A(R(i))[j]$
        Else return "No solution";
    Return the common partition that are determined by the right boundaries;
End of Boundary
```

Lemma 5. *There is an $O(kh)$ time algorithm such that given a (k,h)-rough common partition for S and S' in $\Sigma^n - \Psi_{n,h}(\Sigma)$, it gives a boundary determination for the (k,h)-rough common partition if the boundary determination exists.*

Proof. Assume that string S has rough blocks $S_1, S_2, \ldots, S_k$, and S' has rough blocks $S'_1, S'_2, \ldots, S'_k$. Each S_j corresponds to S'_{i_j} in S'. Assume that $S_i = S[a_i, b_i]$ and $S'_i = S'[a'_i, b'_i]$ for $i = 1, 2, \cdots, k$. According to the Boundary(.), $A(R(j_1)), \cdots, A(R(j_t))$ are disjoint each other and $\cup_{r=1}^{t} A(R(j_r))$ includes all right boundaries of those k blocks $S_1, \cdots, S_k$. This can be found in $O(k)$ time given the rough common partition which contains the correspondence between the blocks in S and those in S'.

$A(R(i_1))[b_{i_1}]$ is a feasible to solution for determining the boundaries in $A(R(i_1))$. It is also possible that we another $j \neq b_{i_1}$ with $A(R_{j_1})[j]$ to be $2h$-consistent. If this happens, we can still extend the partial determination of the boundaries to a determination of all boundaries. Since we have a (k,h)-rough common partition, for each i, we have $end[i] \in (b_i - h, b_i + h)$. Thus the right boundary b_i is in the range $(end[i] - h, end[i] + h)$. After determining $R(i_1)$, all of the boundary points in $A(R(i_1))$ will be fixed. When the boundary point $R(i_1)$ is shifted at most h from its original position b_1, each boundary point in $A(R(i_1))$ is shifted at most h from its original position. The condition that $A(R(i_r))[j]$ is $2h$-consistent in Boundary(.) makes it possible to be extended into the boundaries for k-common partition. After the position of boundary points in $A(R(i_1))$ are fixed, we can keep this process until all boundary points in $A(R(i_1)), \cdots, A(R(i_t))$ are determined. □

Combining the existing modules, we have the following full algorithm for the partition problem.

```
Algorithm
Partition($S, S'$)
Input: two strings $S$ and $S'$.
Steps:
(1)        Check if $S$ and $S'$ are $\Phi_{m,h}(\Sigma)$ string;
(2)        If one of them is in $\Phi_{m,h}(\Sigma)$
           then return "No solution";
(3)        $(k, m, RB)$=RoughBoundaries($S, S'$);
(4)        return Boundary($k, m, RB$);
End of Algorithm
```

Now we give the proof of Theorem 2.

Proof (for Theorem 2). By Lemma 3, it takes $O(nh)$ time to check if a string is in $\Phi_{n,h}(\Sigma)$. By Lemma 4 a rough common partition can be derived in $O(nh^2)$ time. Each position in S is in a local match of length at least $d_0 \log n$. By Lemma 5, the boundaries can be fixed in $O(kh) = O(nh)$ time. Therefore, it takes $O(nh^2)$ time for the entire algorithm. □

Corollary 1. *Let y be an arbitrary positive constant. Assume that the size of alphabet is $t = |\Sigma| \geq 2$. Then there is an $O(\frac{n(\log n)^2}{\log t})$ time algorithm such that the input strings are in $\Sigma^n - \Psi_{n,h}$ and each substring in the partition is of length at least $\frac{d_0 \log n}{\log t}$, it gives a least number k for a common partition solution, where $d_0 = 3c_0$, where c_0 is a constant with $h = c_0 \log n$ that makes $\frac{|\Psi_{n,h}|}{|\Sigma^n|} \leq \frac{1}{2^y}$.*

Proof. It follows from Theorem 2 and Lemma 1. □

Corollary 2. *Let y be an arbitrary positive constant. Assume that the size of alphabet is $t = |\Sigma| \geq 2$. Then there is an $O(\frac{n(\log n)^2}{\log t})$ time algorithm such that one of the input strings in $\Sigma^n - \Psi_{n,h/4}$ and each substring in the partition is of length at least $\frac{d_0 \log n}{\log t}$, it gives a least number common partition solution, where $d_0 = 3c_0$, where c_0 is a constant with $h = c_0 \log n$ that makes $\frac{|\Psi_{n,h}|}{|\Sigma^n|} \leq \frac{1}{2^y}$.*

Proof. If one of the input strings is in $\Sigma^n - \Psi_{n,h/4}$, then the other string is also in $\Sigma^n - \Psi_{n,h}$. Thus, both strings are in $\Sigma^n - \Psi_{n,h}$. It follows from Theorem 2 and Lemma 1. □

4 Algorithms for the Lower Bound of MCSP

We develop two algorithms that give lower bounds on the number of blocks in the minimum common partition solution. One algorithm is based on a greedy approach and the other one is based on the maximum flow over graph. The two algorithms will be presented in the journal version of this paper.

5 Concluding Remarks

In this paper, we present two different algorithms which show the trade off between the computational time and the generality. The polynomial time algorithm can handle most of the cases, but with a requirement of block length. The first algorithm can handle the general case, but takes an exponential time. An interesting problem is to find a fixed parameter tractable (FPT) algorithm for the minimum common string partition problem that runs in time $f(k)n^{O(1)}$, where k is the number of blocks in the partition, and n in the length of two strings, which are of the same length.

Acknowledgments. Bin Fu is supported in part by National Science Foundation Early Career Award 0845376. Binhai Zhu is partially supported by NSF of China under grant 60928006.

References

1. Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., Jiang, T.: Computing the assignment of orthologous genes via genome rearrangement. In: Proc. of the 3rd Asia-Pacific Bioinformatics Conf (APBC 2005), pp. 363–378 (2005)
2. Chrobak, M., Kolman, P., Sgall, J.: The greedy algorithm for the minimum common string partition problem. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) RANDOM 2004 and APPROX 2004. LNCS, vol. 3122, pp. 84–95. Springer, Heidelberg (2004)
3. Cormode, G., Muthukrishnan, S.: The string edit distance matching problem with moves. In: Proc. of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA 2002), pp. 667–676 (2002)
4. Damaschke, P.: Minimum common string partition parameterized. In: Crandall, K.A., Lagergren, J. (eds.) WABI 2008. LNCS (LNBI), vol. 5251, pp. 87–98. Springer, Heidelberg (2008)
5. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Heidelberg (1999)
6. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
8. Goldstein, I., Lewenstein, M.: Quick greedy computation for minimum common string partitions. In: Proc. of the 22nd Annual. Symposium on Combinatorial Pattern Matching (CPM 2011) (to appear 2011)
9. Goldstein, A., Kolman, P., Zheng, J.: Minimum common string partition problem: Hardness and approximations. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 484–495. Springer, Heidelberg (2004)
10. Jiang, H., Zhu, B., Zhu, D., Zhu, H.: Minimum common string partition revisited. J. of Combinatorial Optimization (2010), doi:10.1007/s10878-010-9370-2
11. Kaplan, H., Shafrir, N.: The greedy algorithm for edit distance with moves. Inf. Process. Lett. 97(1), 23–27 (2006)
12. Kolman, P., Walen, T.: Reversal distance for strings with duplicates: Linear time approximation using hitting set. In: Erlebach, T., Kaklamanis, C. (eds.) WAOA 2006. LNCS, vol. 4368, pp. 279–289. Springer, Heidelberg (2007)
13. Kolman, P.: Approximating reversal distance for strings with bounded number of duplicates. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 580–590. Springer, Heidelberg (2005)
14. Kolman, P., Walen, T.: Approximating reversal distance for strings with bounded number of duplicates. Discrete Applied Mathematics 155(3), 327–336 (2007)
15. Shapira, D., Storer, J.: Edit distance with move operations. In: Apostolico, A., Takeda, M. (eds.) CPM 2002. LNCS, vol. 2373, pp. 85–98. Springer, Heidelberg (2002)

Complexity of the Stamp Folding Problem

Takuya Umesato[1], Toshiki Saitoh[2], Ryuhei Uehara[1], and Hiro Ito[3]

[1] School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan
{tbmxcf-ume_sh,uehara}@jaist.ac.jp

[2] ERATO, MINATO Discrete Structure Manipulation System Project, Japan Science and Technology Agency, Hokkaido 060-0814, Japan
t-saitoh@erato.ist.hokudai.ac.jp

[3] Department of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan
itohiro@kuis.kyoto-u.ac.jp

Abstract. For a given mountain-valley pattern of equidistant creases on a long paper strip, there are many folded states consistent with the pattern. Among these folded states, we like to fold a paper so that the number of the paper layers between each pair of hinged paper segments, which is called the crease width at the crease point, is minimized. This problem is called the stamp folding problem and there are two variants of this problem; minimization of the maximum crease width, and minimization of the total crease width. This optimization problem is recently introduced and investigated from the viewpoint of the counting problem. However, its computational complexity is not known. In this paper, we first show that the minimization problem of the maximum crease width is strongly NP-complete. Hence we cannot solve the problem in polynomial time unless P=NP. We next propose an algorithm that solves the minimization problem. The algorithm itself is a straightforward one, but its analysis is not trivial. We show that this algorithm runs in $O\left(n^2\binom{n+k}{k}\right)$ time where n is the number of creases and k is the total crease width. That is, the algorithm runs in $O(n^{k+2})$ time for a constant k. Hence we can solve the problem efficiently for a small constant k.

Keywords: linkage, NP-complete, optimization problem, pleat folding, rigid origami.

1 Introduction

Origamists around the world struggle with the problem for folding an origami model in the best way. Even if you have a good origami model with its crease pattern, this is not the end. That is, they face several problems to search for clever, more accurate, or faster folding sequences and techniques to fold the model. Recently, computer science gives considerable contribution to these problems (see [2]). However, there are still many unsolved problems in this area from the viewpoint of theoretical computer science. We focus here on a simple kind of one-dimensional creasing, where the piece of paper is a long rectangular strip, which can be abstracted into a line segment, and the creases uniformly subdivide the strip. For a given mountain-valley pattern, we aim at folding the long strip into a unit length consistent with the pattern.

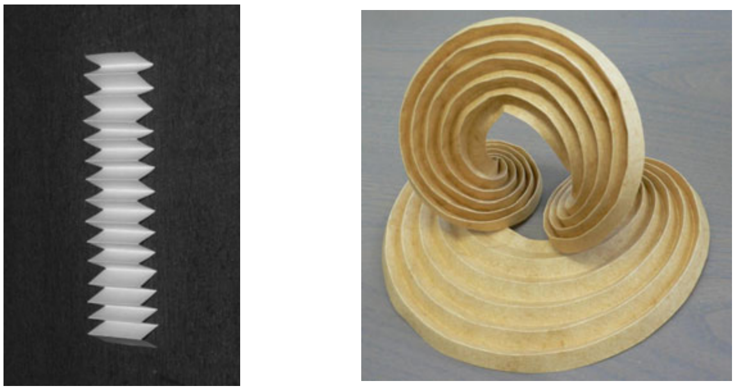


Fig. 1. A simple pleat folding and a curved crease structure folded by Martin Demaine

A mountain-valley pattern is then simply a binary string over the alphabet $\{M, V\}$ (M for mountain, V for valley), which we call a *mountain-valley string*. Of particular interest in origami is the *pleat*, which alternates $MVMVMV\cdots$; see Fig. 1. The pleat folding is unique in the sense that the folded state is unique. That is, there is only one unique folded state consistent with the string, and only the pleat folding has this property (see [10,9]). In general, this is not the case. For example, for a string $MMVMMVMVVVV$, surprisingly, there are 100 distinct folded states consistent with this string. Among them, what is the *best* folded state? From the practical point of view, we like to decrease the number of paper layers between a pair of paper segments hinged at a crease. If we have many paper layers between a hinged pair, it is difficult to fold with accuracy, and if we have too many, we cannot fold any more. This is a typical problem we meet when we fold recent complex origami models.

For a folded state, we define the *crease width* at a crease by the number of the paper layers between the paper hinged at the crease. For example, if the folded state is the pleat, at each crease, the crease width is 0 since we have no paper layers between the paper hinged at the crease. Briefly, we can state the following problem called *stamp folding problem*:

Input: A paper of length $n + 1$ with a mountain-valley string s in $\{M, V\}^n$.
Output: A folded state consistent with s of a small crease width.

From the viewpoint of the optimization, we have two variants of the stamp folding problem:

Problem MinMax**:** To minimize the maximum crease width at a crease in the folded state.
Problem MinTotal**:** To minimize the total crease width for all creases in the folded state.

We note that it is possible to consider minimization problem for the average crease width, but this is equivalent to MinTotal by dividing n. Interestingly, these two problems have different solutions in general. For example, among the 100 folded states for the string *MMVMMVMVVVV*, the minimum maximum crease width is 3, which is achieved by the folded state [4|3|2|5|6|0|1|7|9|11|10|8] (the details of this notation is described later), and the minimum total crease width is 11 by the other state [4|3|2|0|1|5|6|7|9|11|10|8] (see Fig. 2). Moreover, these solutions are unique for this string. (All folded states are checked by an exhaustive search program developed by the third author.)

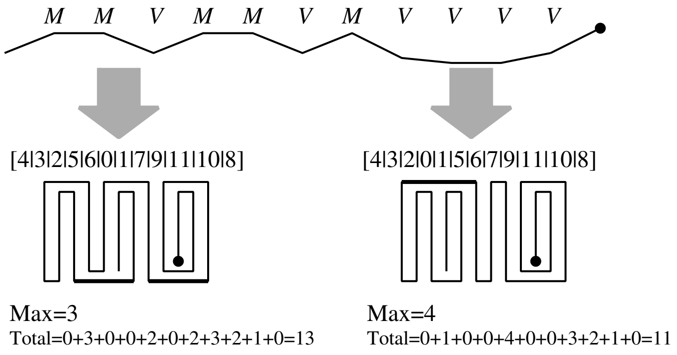


Fig. 2. Two of 100 folded states for the string *MMVMMVMVVVV*. The left folded state achieves the minimum maximum crease width 3, but it has the total crease width 13. The right folded state achieves the minimum total crease width 11, but it has the maximum crease width 4.

The stamp folding problem is introduced by Uehara [10,9] (in the papers, the notion of the crease width was called "stretch"). Uehara introduced the stamp folding problem, showed that it is well-defined even in a simple folding model, and investigated as a counting problem. (The simple folding model is one of basic origami models introduced by Arkin et al. [1].) He proved that the number of the folded states consistent with a random mountain-valley string of length n is exponential on average, but he left its computational complexity open.

In this paper, we first show that the problem MinMax is NP-complete in the strong sense. This solves the open problem in [10,9], and we have no hope to solve the problem efficiently unless P=NP. Hence we aim at solving the problem MinTotal for a bounded k. We then propose a straightforward algorithm; that is, the algorithm tries all possible folded states consistent with a given string s of length n. It is not difficult to see that the algorithm runs in $O(n^2M(n))$ time where $M(n)$ is the number of possible folded states. In [10,9], Uehara showed that $M(n) = \Omega(1.53^n)$ on average. We show that this algorithm runs in $O\left(n^2\binom{n+k}{k}\right)$ time. Hence it runs in $O(n^{k+2})$ time for a small constant k, and we can solve the problem MinTotal in polynomial time.

2 Preliminaries

A *paper strip* is a one-dimensional line segment with *creases* at every integer position. The paper strip is rigid except the creases; that is, we are allowed to fold only at these

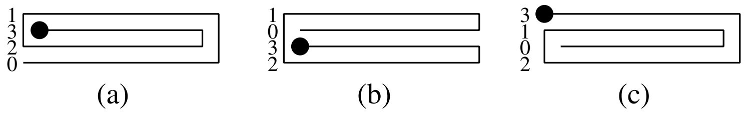


Fig. 3. Three foldings for the mountain-valley string VVV

creases on integer positions. We are given a paper strip of length $n+1$ with a *mountain-valley string* s in $\{M, V\}^n$. At first, the paper strip of length $n+1$ is placed at the interval $[0..n+1]$. (We will refer to this state as an *initial state*.) We aim at obtaining a *folded state* of unit length such that the direction at each crease follows the mountain-valley string s. That is, the ith letter of s indicates the final folded state at the crease i on the integer point i in $[1..n]$. We ignore the thickness of paper, and hence for any mountain-valley string, there exists at least one folded state of unit length.

We call each paper segment between i and $i+1$ at the initial state the *ith segment*. Then each (final) folded state of unit length can be represented by an ordering of the segments; for example, a pleat folding $MVMV$ is described by [0|1|2|3|4] or [4|3|2|1|0], and a mountain-valley string VVV produces [1|3|2|0], [1|0|3|2], [3|1|0|2], or their reverses (Fig. 3). We distinguish between the left and right ends of the paper strip, but we sometimes identify one folded state and its reverse since they are essentially the same. In fact, the side of a folded state is sometimes changed when we fold all paper layers at a crease from right to left or from left to right.

For a mountain-valley string s, we call a folded state *legal for s* if it follows the string. That is, for example, the mountain-valley string VVV has three different legal folded states [1|3|2|0], [1|0|3|2] and [3|1|0|2]. A mountain-valley string $MVMVMV\cdots$ is called a *pleat*. It is known that the legal folded state is unique (up to reversal of the paper) if and only if s is a pleat [10,9].

In a folded state, a crease or a segment is *visible* if it is not covered by the other paper. We note that in a folded state of unit length, there exist exactly two visible segments (if $n \geq 1$), while there can be many visible creases on both sides. For example, the folded state Fig. 3(a) has two visible creases 1 and 2, while the folded state Fig. 3(b) has three visible creases 1, 2, and 3.

3 NP-Completeness

In this section, we show NP-completeness of the problem MinMax. Thus, in this section, *the crease width of a folded state* is defined by the maximum crease width at each crease. We remind that the pleat folding is unique in the sense that the folded state is unique [10,9], and it has the crease width 0. We give other characteristic and useful patterns at first:

Observation 1. *Let s be a mountain-valley string V^n (or M^n) for a positive integer n. Then the number of legal folded states for s is n, and the crease width of each state is* $n-1$ *(see Fig. 3).*

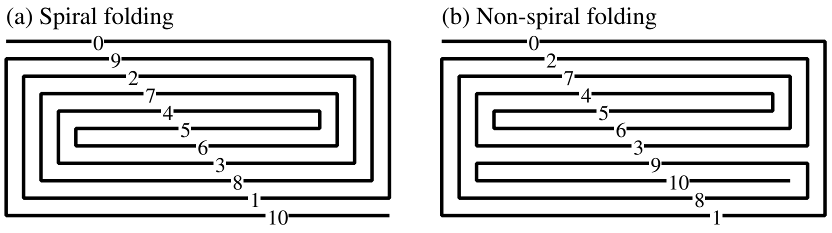


Fig. 4. Spiral folding and non-spiral folding

Observation 2. *Let s be a mountain-valley string M^nV^n for a positive integer n. Then the number of legal folded states for s is n^2. Among them, if the* 0*th segment and* 2*nth segment are visible, the legal folded state is* $[0|2n-1|2|2n-3|\ldots|2i|2(n-i)-1|\ldots|1|2n]$ *or its reversal.*

Two legal folded states of M^5V^5 can be found in Fig. 4. In Fig. 4(a), the 0th and 10th segments are visible as stated in Observation 2. We call this folded state *spiral folding* of length $2n$. In the other legal folded states like Fig. 4(b), either the 0th and the 1st segments are visible, or the $(2n-1)$st and the $2n$th segments are visible.

Now we are ready to show the complexity of the following problem:

MinMax
INSTANCE: A mountain-valley string $s \in \{M, V\}^n$ and a natural number k.
QUESTION: Is there a legal folded state for s of crease width at most k?

Theorem 3. MinMax *is NP-complete.*

Proof. It is easy to see that the problem is in NP. We in the following show the hardness by reducing from 3-PARTITION, defined as follows.

3-PARTITION (cf. [5])
INSTANCE: A finite set $A = \{a_1, a_2, \ldots, a_{3m}\}$ of $3m$ elements in $\mathbb{Z}_+$ and a bound $B \in \mathbb{Z}_+$ such that each a_j satisfies $B/4 < a_j < B/2$ and $\sum_{j=1}^{3m} a_j = mB$.
QUESTION: Can A be partitioned into m disjoint sets $A^{(1)}, A^{(2)}, \ldots, A^{(m)}$ such that $\sum_{a_j \in A^{(i)}} a_j = B$ for $1 \le i \le m$?

It is well-known that 3-PARTITION is strongly NP-complete, meaning that it is NP-hard even if the input is written in unary notation [5]. To begin with, we describe a construction of an instance s and k of MinMax for a given instance $a_1, \ldots, a_{3m}$ and B of 3-PARTITION.

Construction of a mountain-valley string s and an integer k. For each individual element $a_j \in A$, we first construct the following string x_j.

$$x_j = \begin{cases} V^{a_j m^3} M^{a_j m^3} & \text{if } j \text{ is odd,} \\ M^{a_j m^3} V^{a_j m^3} & \text{otherwise.} \end{cases} \tag{1}$$

Let p_i be the string $(MV)^i$, and p'_i be the string $(VM)^i$. We next define the following substring s_j of s for each $j = 1, 2, \ldots, 3m$.

$$s_j = \begin{cases} p'_m x_j p'_m & \text{if } j \text{ is odd,} \\ p_m x_j p_m & \text{otherwise.} \end{cases} \tag{2}$$

We also define two strings $t_1 = M^{2Bm^3+16m^2}$, $t_2 = V^{Bm^3+8m^2+1} M^{Bm^3+8m^2}$ and another string $f = p_{m+1}$. We call t_1 and t_2 *terminators*. Then, we concatenate all of them and obtain the mountain-valley string $s = t_1 f t_2 s_1 s_2 \ldots s_{3m}$. We also let $k = 2Bm^3 + 16m^2$. Clearly, this instance can be constructed in polynomial time from the 3-PARTITION instance.

Roughly speaking, the ith valley of f corresponds to the set $A^{(i)}$ and it will catch the paper layers in three x_j's that make crease width (at most) k in total. We call the ith valley of f the ith *folder*.

We here state two lemmas that completes the proof of Theorem 3.

Lemma 1. *If the instance $\{a_1, a_2, \ldots, a_{3m}\}$ and B of 3-PARTITION has a solution, the paper strip with the mountain-valley string s has a legal folded state of crease width at most k.*

Proof. Suppose a partition $A^{(1)}, \ldots, A^{(m)}$ of A is a solution of 3-PARTITION. Then we have $A^{(i)} \subset A$, $\left|A^{(i)}\right| = 3$, $\sum_{a_j \in A^{(i)}} a_j = B$ for each i in $\{1, \ldots, m\}$, and $A = \dot{\bigcup}_{i=1}^{m} A^{(i)}$. From the partition, we construct a legal folded state of the paper strip of crease width k.

We first consider the legal folding of the substring $s' = t_1 f t_2$. By Observation 1, the number of legal folded states of t_1 is $k+1$. Among them, only one folded state can have an "edge" of t_1 outside that is connected to the part of f. If we fold the edge of t_1 inside, the all paper layers for $f t_2$ are wrapped by the paper layers for t_1 and the maximum crease width cannot be bounded above by k. Therefore, to achieve the maximum crease width at most k, we have to fold t_1 as $[k|k-2|\ldots|k-2i|\ldots|2|0|1|3|\ldots|2i+1|\ldots|k-1]$. The legal folded state of the pleat f is unique, and hence the pleat f is folded into $[k+1|k+2|\ldots|k+m+2]$. Next, we turn to the terminator t_2. The legal folded state for t_2 up to crease width k can be only achieved by the spiral folding $[k+m+3|2k+m+3|k+m+5|2k+m+1|\ldots|2k+m+2|k+m+2]$, and it is put into the $(m+1)$st folder.

So far, we fold the substring s' as follows (see Fig. 5).

$$\begin{gathered} [k|k-2|\ldots|k-2i|\ldots|2|0|1|3|\ldots|2i+1|\ldots|k-1 \\ |k+1|k+2|\ldots|k+m \\ |k+m+1|2k+m+3|k+m+3|2k+m+1|\ldots|2k+m+2|k+m+2]. \end{gathered}$$

For each $A^{(i)} = \{a_j, a_{j'}, a_{j''}\}$, we fold x_j, $x_{j'}$, and $x_{j''}$ and put them together into the ith folder, intuitively. First suppose j is odd. We fold the paper of the substring x_j into the ith folder. This folding consists of three steps (Fig. 6). First, we fold the paper of the first substring p'_m in s_j as follows. For each mountain M in p'_m, we put it into each folder from the mth folder in descending order. When it reaches the ith folder, the remaining segments of the p'_m are put into the ith folder.

Then the substring x_j is folded into spiral in the ith folder. More precisely, we fold the paper of x_j in spiral folding to the folded state $[t|t + 2a_j m^3 - 1|t + 2|t + 2a_j m^3 -$

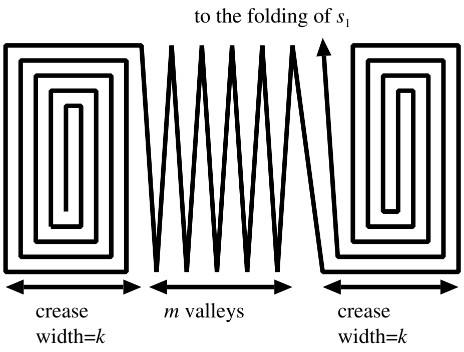


Fig. 5. Folding of a pleat f between two terminators t_1 and t_2

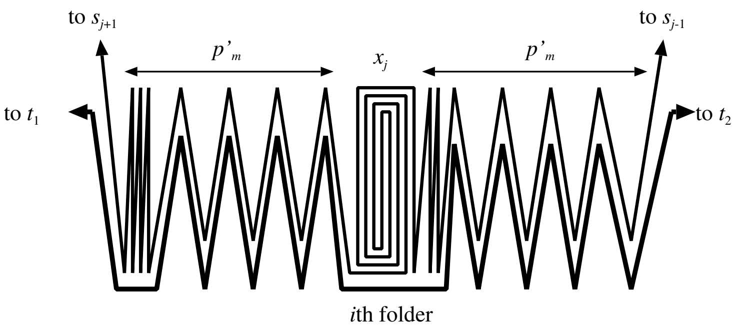


Fig. 6. Folding of $s_j = p'_m x_j p'_m$

$3|\dots|t+2l|t+2(a_jm^3-l)-1|\dots|t+1|t+2a_jm^3]$, where a paper of x_j corresponds to an interval $[t..t+2a_jm^3+1]$. The folded state of x_j is put into the ith folder.

For each mountain M of the second p'_m, we put it in the folder from the ith folder in descending order. When it reaches to the 1st folder, all remaining pleats are put into the 1st folder.

For even j, the substring s_j can be folded in the reverse way.

We show that the maximum crease width of this legal folded state is at most k. The maximum crease width in the t_1 is k between the segments k and $k-1$. The maximum crease width in the t_2 is also k between the segments $k+m+3$ and $k+m+2$. At each ith folder, we have put the paper of strings x_j, $x_{j'}$, and $x_{j''}$, and some segments of p_m or p'_m for each $h \in \{1,\dots,3m\}$. Therefore, the maximum crease width at the ith folder is at most $2(a_jm^3+a_{j'}m^3+a_{j''}m^3)+2\cdot2\cdot m\cdot 3m = 2Bm^3+12m^2 < k$. □ (of Lemma 1)

Lemma 2. *If the paper strip with the mountain-valley string s has a legal folded state of crease width at most k, the instance* $\{a_1,a_2,\dots,a_{3m}\}$ *and B of 3-PARTITION has a solution.*

Proof. We can observe that the folded state has to satisfy the following conditions to achieve the crease width (at most) k: (1) $s' = t_1 f t_2$ has the unique folding state in Fig. 5, (2) x_j should be put into some folder, and (3) each folder binds paper layers corresponding to exactly three x_js.

The condition (1) is easy to see by Observations 1 and 2. (We then have the maximum crease width exactly k.) Then the condition (2) follows (1); otherwise, some paper layers of s_j has to stride over one of terminators, and then we have the crease width greater than k at the crease. Now we turn to the condition (3). To derive a contradiction, we suppose some folder contains paper layers corresponding to at most 2 x_js. Then, by the pigeon hole principle with the condition (2), we have some other folder that contains paper layers corresponding to at least 4 x_js, say $x_{j_1}, x_{j_2}, x_{j_3}, x_{j_4}$ of crease width at most k at the folder. Then, we have $a_{j_1} + a_{j_2} + a_{j_3} + a_{j_4} \leq B$. However, by the assumption of the instance of 3-PARTITION, $a_{j_1} + a_{j_2} + a_{j_3} + a_{j_4} > 4(B/4) = B$, which is a contradiction. Therefore we have the condition (3).

By the conditions, we can certainly reconstruct a solution of the instance of 3-PARTITION from the legal folded state for the string s with the maximum crease width k. □ (of Lemma 2)

Now we turn to the proof of Theorem 3. By Lemmas 1 and 2, the resultant paper strip has a legal folded state of maximum crease width k if and only if the instance of 3-PARTITION has a solution. Hence we have the theorem. □ (of Theorem 3)

4 Tractability for Bounded *k*

In this section, we show that the problem MinTotal is tractable for a small constant k. That is, in this section, *the crease width of a folded state* is defined by the total crease width of the crease width at each crease. We are given a mountain-valley string s and a upper bound k of the total crease width, and we design an algorithm that determines if there exists a legal folded state for s of total crease width at most k. The algorithm is a simple and straightforward one that tries all possible folding ways (see Algorithm 1).

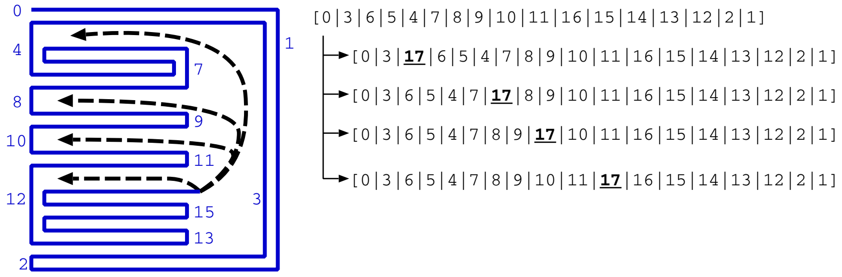
```
Input  : $s \in \{M, v\}^n$ with $n \geq 1$, an integer $k$
Output: All possible folded states $P$ consistent with $s$ of crease width at most $k$
1  $P$ is initialized by the 0th segment laid on the interval $[0, 1]$;
2  fold($P, 1, k$);
   // attach the 1st segment to the current $P$ with constraint crease
      width $\leq k$.
```

Algorithm 1. Enumeration Algorithm

The algorithm indeed outputs all possible legal folded states for s. That is, if a given input has a solution, we will have at least one feasible folded state. The procedure fold(P, i, k) is crucial (see Procedure fold(P, i, k)).

```
if $i = n$ then output $P$ else
    pick up the $i$th segment;
    foreach layer of the current $P$ do
        if $i$th segment can be inserted to the layer then
            update $P$ by inserting the $i$th segment into the layer on the interval $[0, 1]$;
            let $k'$ be the crease width at the crease $i$;
            let $j$ be the number of creases whose crease widths are increased by the
            insertion of the $i$th segment;
            // We should have $k - (k' + j) \geq 0$ since this is a possible
               layer.
            fold($P, i + 1, k - (k' + j)$);
```

Procedure `fold`(P, i, k)



```
[0|3|6|5|4|7|8|9|10|11|16|15|14|13|12|2|1]
  ├─►[0|3|17|6|5|4|7|8|9|10|11|16|15|14|13|12|2|1]
  ├─►[0|3|6|5|4|7|17|8|9|10|11|16|15|14|13|12|2|1]
  ├─►[0|3|6|5|4|7|8|9|17|10|11|16|15|14|13|12|2|1]
  └─►[0|3|6|5|4|7|8|9|10|11|17|16|15|14|13|12|2|1]
```

Fig. 7. Possible choices in `fold`$(P, 17, k)$

Example 1. The current folded state is [0|3|6|5|4|7|8|9|10|11|16|15|14|13|12|2|1] in Fig. 7. Dotted lines indicate all possible layers in `fold`$(P, 17, k)$ (for some large k). In any case, the crease widths at the crease 1 and 3 are increased. On the other hand, the crease width at one of the 4, 8, 10 and 12 is increased according to the chosen layer. The crease width at 17 will be 8, 4, 2, or 0, respectively.

The correctness of the algorithm is clear. Let $M(s, k)$ be the number of feasible folded states consistent to s of total crease width exactly k. Then the running time of the algorithm is $O(n^2 \sum_{i=0}^{k} M(s, i))$, where $n = |s|$. (Note that we can improve this bound to $O(n \sum_{i=0}^{k} M(s, i))$ with sophisticated data structure. But we omit the details in this paper since it is not essential from the theoretical point of view.) Thus we estimate $M(s, k)$ as follows:

Lemma 3. *Let s be a mountain-valley string of length n, and k be an integer. Then* $M(s, k) \leq \binom{n+k-1}{k}$.

Proof. Let P be a legal folded state for s of crease width exactly k. Let c_i be the crease width at ith crease with $1 \leq i < n$. Then, an upper bound of $M(s, k)$ is all combinations of c_is with $\sum_{i=1}^{n} c_i = k$ and $c_i \geq 0$. This is one of the classic ball-bin problems of placing

k balls into n bins. This observation gives us an upper bound $\binom{n+k-1}{k}$ of $M(s,k)$ (see, e.g., [3, Chapter II. 5]). □

By Lemma 3, we have the following theorem:

Theorem 4. *Algorithm 1 solves* MinTotal *in* $O\left(n^2\binom{n+k}{k}\right)$ *time. That is, it solves* MinTotal *in* $O(n^{2+k})$ *when k is a positive constant.*

Proof. $M(s,i)$ is an increasing function for i. Here using the combinatorial identity (see, e.g., [8, Eq. (7) in page 217])

$$\binom{n}{0} + \binom{n+1}{1} + \cdots + \binom{n+r}{r} = \binom{n+r+1}{r}$$

we have $\sum_{i=0}^{k} M(s,i) \le \sum_{i=0}^{k} \binom{n+i-1}{i} = \binom{n+k}{k}$. Since $\binom{n}{k} \le (en/k)^k$ (see, e.g., [6]), the theorem follows. □

5 Concluding Remarks

In this paper, we show that the minimization problem of the maximum crease width is strongly NP-complete. We furthermore show that a brute force algorithm that solves the minimization problem of the total crease width in $O\left(n^2\binom{n+k}{k}\right)$ time, where k is the total crease width. That is, we can solve the minimization problem of the total crease width in $O(n^{k+2})$ time if k is a constant. We left the problem to determine if the minimization problems are fixed parameter tractable (FPT) or not [4,7]. Especially, is there any algorithm that solves the minimization problem of the total crease width in time $O(f(k) \cdot n^c)$ for some constant c, making it FPT?

Acknowledgement

The authors are grateful to Yoshio Okamoto and Shuji Kijima for their helpful comments. The third author also thanks Martin Demaine, who folded the curved crease structure in Fig. 1.

References

1. Arkin, E.M., Bender, M.A., Demaine, E.D., Demaine, M.L., Mitchell, J.S.B., Sethia, S., Skiena, S.S.: When can you fold a map? Comput. Geom.Theory Appl. 29(1), 23–46 (2004)
2. Demaine, E.D., O'Rourke, J.: Geometric Folding Algorithms: Linkages, Origami, Polyhedra. Cambridge University Press, Cambridge (2007)
3. Feller, W.: An Introduction to Probability Theory and Its Applications, 3rd edn., vol. 1. John Wiley & Sons, Inc., Chichester (1968)
4. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability — A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
6. Motwani, R., Raghavan, P.: Randomized Algorithms, Cambridge (1995)

7. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms, Oxford (2006)
8. Tucker, A.: Applied Combinatorics, 3rd edn. John Wiley & Sons, Inc., Chichester (1995)
9. Uehara, R.: On Stretch Minimization Problem on Unit Strip Paper. In: 22nd Canadian Conference on Computational Geometry (CCCG), pp. 223–226 (2010)
10. Uehara, R.: Stretch Minimization Problem of a Strip Paper. In 5th International Conference on Origami in Science, Mathematics and Education (5OSME) (2010)

On the Number of Solutions of the Discretizable Molecular Distance Geometry Problem

Leo Liberti[1], Benoît Masson[2], Jon Lee [3],
Carlile Lavor [4], and Antonio Mucherino[5]

[1] LIX, École Polytechnique, 91128 Palaiseau, France
liberti@lix.polytechnique.fr
[2] IRISA, INRIA, Campus de Beaulieu, 35042 Rennes, France
benoit.masson@inria.fr
[3] Dept. of Mathematical Sciences, IBM T.J. Watson Research Center, PO Box 218, Yorktown Heights, NY 10598, USA
jonlee@us.ibm.com
[4] Department of Applied Mathematics (IMECC-UNICAMP), State University of Campinas, 13081-970, Campinas - SP, Brazil
clavor@ime.unicamp.br
[5] CERFACS, Toulouse, France
mucherino@cerfacs.fr

Abstract. The Discretizable Molecular Distance Geometry Problem is a subset of instances of the distance geometry problem that can be solved by a combinatorial algorithm called "Branch-and-Prune". It was observed empirically that the number of solutions of YES instances is always a power of two. We perform an extensive theoretical analysis of the number of solutions for these instances and we prove that this number is a power of two with probability one.

Keywords: distance geometry, symmetry, Branch-and-Prune, power of two.

1 Introduction

We consider the following problem arising in the analysis of Nuclear Magnetic Resonance (NMR) data for general molecules.

Molecular Distance Geometry Problem (MDGP).
Given a simple undirected graph $G = (V, E)$ and a function $d : E \to \mathbb{R}$, decide whether there is an embedding $x : V \to \mathbb{R}^3$ such that

$$\forall \{u, v\} \in E \quad (||x_u - x_v|| = d_{uv}) \tag{1}$$

The MDGP is a mixed-combinatorial optimization problem; it can be cast as the global optimization problem $\min \sum_{\{u,v\} \in E} (||x_u - x_v||^2 - d_{uv}^2)^2$ in continuous variables, which is generally solved using continuous search techniques [1,2]. The generalization of the MDGP to arbitrary dimensions asks for an embedding of G in $\mathbb{R}^K$ satisfying (1) and is called the DISTANCE GEOMETRY PROBLEM (DGP). The DGP is strongly **NP**-hard [3]; it is related to the Euclidean Distance Matrix Completion Problem (EDMCP) [4] (whose complexity status is currently unknown), the difference being that in the EDMCP the dimension K of the embedding space is part of the output rather than part of the input.

Finding a Euclidean embedding of a weighted graph has two main applications: to molecular conformation [5] and to sensor networks [6,7]. The results of this paper were inspired by the application to the conformation of proteins: in particular, chemical analysis and NMR experiments can help identify a subset of inter-atomic distances [8]. The motivation is that the function of a protein is determined by its 3D structure [9]. Since proteins are a strict subset of molecules, it makes sense to ask whether there the restriction of the MDGP to proteins might yield more efficient methods than those developed for the MDGP applied to general molecules. In 2005 two of the authors of this paper (CL and LL) started working on a discrete algorithm which exploits two observations: (i) proteins are organized in a *backbone* and some *side chains*, which can be embedded separately, once the backbone embedding is known [10]; (ii) the distances between any atom v of the backbones, seen as as a total order on the set of atoms, to its three immediate predecessors (i.e., $v-1, v-2, v-3$) are generally known (and by applying certain technical devices to the order can be assumed to be precise [11]). This algorithm, called Branch-and-Prune (BP), is based on the *Sphere Intersection Property* (SIP): the intersection of K spheres in $\mathbb{R}^K$ generally consists of either 0 or 2 points. Here the term *generally* has a definite significance: it means that the set K-tuples of spheres for which the SIP does not hold has Lebesgue measure 0 in the set of all possible K-tuples of spheres.

In the following, we identify atoms with the set V of vertices of a given graph G, whose edge set E includes the pairs of atom for which a distance is known. The weight of each edge $\{u, v\} \in E$ is the value of the distance d_{uv}, and an order on the vertices (the backbone order in the case of proteins) is given. BP exploits the SIP by performing a binary search in the space of embeddings: under the hypothesis that *for each vertex of rank $> K$ in the order, the distances to its K immediate predecessors are known*, the BP places a vertex v in both of the positions guaranteed by the SIP, verifies whether these are compatible with the distances to *all* adjacent predecessors of v, and then accordingly recurses the search to the successor of v. This yields a worst-case exponential behaviour, occurring when the set of adjacent predecessors of each vertex v is equal to the set of its K immediate predecessors. In practice, however, the BP outperforms its continuous search competitors in both efficiency and reliability [12]. One particularly useful feature of BP is that, because the search is complete, it finds the set X of all incongruent embeddings for a given graph. In a sequence of papers (the main ones being [13,12,14,15,16,17,18]) we developed this idea in a number of directions. In particular, we defined a new optimization problem, the DISCRETIZABLE MDGP (DMDGP) [12] as the class of all DGP instances that satisfy the conditions required by the BP: the existence of a vertex order such that the K immediate predecessors of each vertex v of rank $> K$ are adjacent to v in G, and the fact that d satisfies strict simplex inequalities [19,15].

In all our computational tests on DMDGP instances, we observed that the number of incongruent embeddings is a power of two: this comes to no surprise in the exponential worst case mentioned above, but there is no apparent reason why this should be the case when adjacent predecessors also include other vertices than the K immediate predecessors (and, indeed, in Sect. 6 we exhibit a set of counterexamples to the conjecture that for all YES instances of the DMDGP $\exists h \in \mathbb{N}\ (|X| = 2^h)$). Yet, the computational trend remained unexplained. The contribution of this paper is a proof that the set of YES instances of the DMDGP such that $|X|$ is a power of two has Lebesgue measure 1 in the set of all YES instances of the DMDGP. The statement is based on the assumption that we consider solutions (i.e. graph embeddings) whose components range in the uncountable set $\mathbb{R}^K$. Our result is nontrivial, and accordingly the proof, which consists of several lemmata, propositions and theorems, is long, technical and difficult: because of the page limit, all proofs are in the appendix. The result is nonetheless very important insofar as it explains the behaviour of a practically useful solution method.

The rest of this paper is organized as follows. We give a formal description of the DMDGP in arbitrary dimensions (Sect. 2) and of the BP algorithm and some of its theoretical properties (Sect. 3); we then study some geometrical aspects of the BP tree (Sect. 4), and prove that the number of solutions of YES instances of the DMDGP is a power of two with probability one (Sect. 5). We exhibit a (zero measure) family of counterexamples to the "power of two" conjecture in Sect. 6.

2 The Formal Definition of the Discretizable Molecular Distance Geometry Problem

For a set $U = \{x_i \in \mathbb{R}^K \mid i \leq K+1\}$ of points in $\mathbb{R}^K$, let D be the symmetric matrix whose (i,j)-th component is $\|x_i - x_j\|^2$ for all $i,j \leq K+1$ and let D' be D bordered by a left $(0,1,\ldots,1)^\top$ column and a top $(0,1,\ldots,1)$ row (both of size $K+2$). Then the Cayley-Menger formula states that the volume $\Delta_K(U)$ of the K-simplex on U is given by $\Delta_K(U) = \sqrt{\frac{(-1)^{K+1}}{2^K(K!)^2}|D'|}$. The strict simplex inequalities are given by $\Delta_K(U) > 0$. For $K = 3$, these reduce to strict triangle inequalities. We remark that only the distances of the simplex edges are necessary to compute $\Delta_K(U)$, rather than the actual points in U; the needed information can be encoded as a complete graph $\mathbf{K}_{K+1}$ on $K+1$ vertices with edge weights as the distances.

Let $n = |V|$ and $m = |E|$. For all $v \in V$, let $N(v) = \{u \in V \mid \{u,v\} \in E\}$ be the star of vertices around v (also called the adjacencies of v); for a directed graphs (V,A), where $A \subseteq V \times V$, we denote the outgoing star by $N^+(v) = \{u \in V \mid (v,u) \in A\}$. For an order $<$ on V, let $\gamma(v) = \{u \in V \mid u < v\}$ be the set of predecessors of v, and let $\rho(v) = |\gamma(v)| + 1$ be the rank of v in $<$. For $V' \subseteq V$, we denote by $G[V']$ the subgraph of G induced by V'. For a finite set M, let $\mathcal{P}(M)$ be its power set. We call an embedding x of G *valid* if (1) holds for G. For a sequence $x = (x_1, \ldots, x_n)$ and a subset $U \subseteq \{1, \ldots, n\}$ we let $x[U]$ be the subsequence of x indexed by U. If x is an initial subsequence of y, then y is an *extension* of x. For each $v \in V$ with $\rho(v) > K$ we let U_v be the set of the K immediate predecessors of v, and remark that $U_v \subseteq N(v) \cap \gamma(v)$.

The Generalized DMDGP. Given an undirected graph $G = (V, E)$, an edge weight function $d : E \to \mathbb{R}_+$, an integer $K > 0$, a subset $V_0 \subseteq V$ with $|V_0| = K$, a partial embedding $\bar{x} : V_0 \to \mathbb{R}^K$ valid for $G[V_0]$, and a total order $<$ on V such that:

$$\{v \in V \mid \rho(v) \leq K\} = V_0; \tag{2}$$

$$\forall v \in V \quad (\rho(v) > K \to |N(v) \cap \gamma(v)| \geq K); \tag{3}$$

$$\forall v \in V \smallsetminus V_0 \ (G[U_v] = \mathbf{K}_K \wedge \varDelta_{K-1}(U_v) > 0), \tag{4}$$

decide whether there is a valid extension $x : V \to \mathbb{R}^K$ of $\bar{x}$.

Conditions (2-4) allow the search for the Euclidean position of vertex v to only depend on the K vertices of rank preceding $\rho(v)$, as x_v is the intersection of at least K spheres centered at x_u and with radius d_{uv} for all $u \in N(v) \cap \gamma(v)$. This, in particular, implies that the predecessors of v are placed before v, so that all of the distances between all predecessors are known when placing v. Thus, we can also solve instances for which $G[U_v]$ is not the full K-clique, although they are not formally in the generalized DMDGP.

We remark that the SIP is independent of U_v, so that we could simply replace U_v with any subset of $N(v) \cap \gamma(v)$ with cardinality K. This actually yields a larger instance set called Discretizable Distance Geometry Problem (DDGP), or DDGP$_K$ if K is fixed and not part of the input, discussed in [14]. We shall see, however, that the assumption that U_v contains the K *immediate* predecessors of v will be crucial in the following (this, by the way, also explains why the generalized DMDGP is not called "DDGP" in analogy with MDGP→DGP). In the rest of the paper we use the acronym DMDGP to actually mean the *generalized* DMDGP, and we use the name DMDGP$_3$ to name the original DMDGP in $\mathbb{R}^3$. Complexity-wise, a polynomial reduction from Subset-Sum to the DMDGP$_3$ [12] shows that the DMDGP is **NP**-hard.

3 Sphere Intersections and Reflections

The BP algorithm for the DMDGP$_3$, presented in [13], can easily be extended to the DMDGP. As mentioned above, once the vertices of U_v have been embedded in $\mathbb{R}^K$, the known distances from vertices in U_v to a given v will enforce the position of v as the intersection of K spheres. Under strict simplex inequalities, this intersection consists of at most two distinct points. The BP exploits this fact to recursively generate a binary search tree of height at most n where a node at level i represents a possible placement in $\mathbb{R}^K$ of the vertex of G with rank i in $<$. Paths of length n correspond to valid embeddings.

Let G be a DMDGP instance. Consider $v \in V$ with rank $\rho(v) = i > K$, let $G^v = G[\gamma(v) \cup \{v\}]$ and x be a valid embedding of $G[\gamma(v)]$. We characterize the number of extensions of x valid for G^v in the following lemmata (which also hold for the DDGP). Lemmata 3.1 and 3.2 essentially state that $G[\{v\} \cup (N(v) \cap \gamma(v))]$ are rigid and, respectively, uniquely rigid graphs.

In the following, we assume that the probability of any point of $\mathbb{R}^K$ belonging to any given subset of $\mathbb{R}^K$ having Lebesgue measure zero is equal to zero. Based on this assumption, when we state "$(\forall p \in P \; F(p))$ with probability 1" for a certain well-formed formula F with a free variable ranging over an uncountable set P, we really mean that there exists a Lebesgue measurable subset $Q \subseteq P$ with Lebesgue measure 1 in P such that $\forall q \in Q \; F(p)$. For example, the statement of Lemma 3.1 should be read as follows: the set of DMDGP instances and partial embeddings x for which the result does not hold has Lebesgue measure 0 in the set of all DMDGP instances and partial embeddings. We remark that this is different from the usual genericity notion employed in rigidity theory [20], which requires distances to be algebraically independent over $\mathbb{Z}$. Since our instances come from experimental measurements over existing structures, the distances may not be independent. One consequence is the validity of Lemma 3.2, which would not hold with the stronger genericity requirement (the intersection of $K+1$ "generic spheres" in $\mathbb{R}^K$ is empty).

Lemma 3.1. *If $|N(v) \cap \gamma(v)| = K$ then there are at most two distinct extensions of x that are valid for G^v. If one valid extension exists, then with probability 1 there are exactly two distinct valid extensions.*

Lemma 3.2. *If $|N(v) \cap \gamma(v)| > K$ then, with probability* 1*, there is at most one extension of x.*

Lemma 3.3. *With the notation of Lemma 3.1, if $\bar{x}$ is a valid embedding for $G[U_v]$, then z'' is a reflection of z' with respect to the hyperplane through the K points of $\bar{x}$.*

Reflections with respect to hyperplanes are isometries, and can therefore be represented by linear operators. If $a \in \mathbb{R}^K$ is the unit normal vector to a hyperplane H containing the origin, then the reflection operator R_0 w.r.t. H can be expressed in function of the standard basis by the matrix $I - 2aa^\top$, where I is the $K \times K$ identity matrix [21]. Let H be a hyperplane with equation $a^\top x = a_0$ (with $a_0 \neq 0$) and a_i, for some $1 \leq i \leq K$, be the nonzero coefficient of smallest index in a. Then, the reflection operator R acting on a point $p \in \mathbb{R}^K$ w.r.t. H is given by $R(p) = R_0(p - \frac{a_0}{a_i} e_i) + \frac{a_0}{a_i} e_i$, where $e_i \in \mathbb{R}^K$ is the unit vector with 1 at index i and 0 elsewhere: we first we translate p so that we can reflect it using R_0 w.r.t. the translation of H containing the origin, then we perform the inverse translation of the reflection.

3.1 Branch-and-Prune

A formal description of the BP algorithm for the DMDGP is given in Alg. 1. It builds a binary search tree $\mathcal{T} = (\mathcal{V}, \mathcal{A})$, directed from the root to the leaves, whose nodes are triplets $\alpha = (x(\alpha), \lambda(\alpha), \mu(\alpha))$. For $\alpha \in \mathcal{T}$ we denote by $\mathsf{p}(\alpha)$ the unique path from the root node r of $\mathcal{T}$ to α; $x(\alpha)$ is an extension of the embedding x^- found on $\mathsf{p}(\alpha^-)$, where α^- is the unique parent node of α. The symbol $\lambda(\alpha) \in \{0, 1\}$ distinguishes whether α is a "left" or a "right" subnode of α^-. More precisely, let α be a node at level i in $\mathcal{T}$, $v = \rho^{-1}(i)$, $\bar{x}$ be a partial embedding of $G[U_v]$, and $a_v^\top x = a_{v0}$ be the equation of the ($(K-1)$-dimensional by (4)) hyperplane through the points of $\bar{x}$. Assuming $u = \rho^{-1}(i-1)$, $a_v \in \mathbb{R}^K$ is oriented so that $a_v \cdot a_u \geq 0$; then:

$$\lambda(\alpha) = \begin{cases} 0 & \text{if } a_v^\top x(\alpha)_i \leq a_{v0} \\ 1 & \text{if } a_v^\top x(\alpha)_i > a_{v0}. \end{cases} \tag{5}$$

Lastly, $\mu(\alpha) = \boxplus$ if x is a valid extension of x^-, in which case the node is said to be *feasible*, and $\mu = \boxminus$ otherwise. This allows us to retrieve the set X of all valid embeddings of G by simply traversing $\mathcal{T}$ backwards from the leaf nodes marked $\boxplus$ up to r.

We remark that Alg. 1 differs from the original BP formulation [13] because it applies to K dimensions and explicitly stores several details of the binary search tree.

Lemma 3.4. *At termination of Alg. 1, X contains all valid embeddings of G extending $\bar{x}$.*

We now partition $\mathcal{V}$ in pairwise disjoint subsets $\mathcal{V}_1, \ldots, \mathcal{V}_n$ where for all $i \leq n$ the set $\mathcal{V}_i$ contains all the nodes of $\mathcal{V}$ at level i of the tree $\mathcal{T}$.

Proposition 3.5. *With probability 1, there is no level $i \leq n$ having two distinct feasible nodes $\beta, \theta \in \mathcal{V}_i$ such that $|\{\alpha \in N^+(\beta) \mid \mu(\alpha) = \boxplus\}| = 1$ and $|\{\alpha \in N^+(\theta) \mid \mu(\alpha) = \boxplus\}| = 2$.*

We remark that Prop. 3.5 also holds for the DDGP provided U_v is chosen in Alg. 1 as any subset of $N(v) \cap \gamma(v)$ satisfying the constraints of Eq. (4).

4 Geometry in BP Trees

The most important result of this section is that, for any valid embedding $y \in X$, if the BP tree branches at level $i = \rho(v)$ on the path to y and both branches continue to the last level, then the embedding obtained by reflecting all the points of y past the $(i-1)$-th vertex through the hyperplane defined by $y[U_v]$ is also valid with probability 1. We remark that the results in this section only apply to the DMDGP (not to the DDGP, as shown in the counterexample of Fig. 3).

We need to emphasize those BP branchings which carry on to feasible leaf nodes along both branches. For $y \in X$ and a vertex $v \in V \smallsetminus V_0$ we denote $\Upsilon(y, v)$ the following property:

Algorithm 1. The Branch and Prune algorithm

```
Require: Partial embedding $\bar{x}$ of first $K$ vertices of $G$
Ensure: Set $X$ of valid embeddings of $G$
 1: Let $\alpha = (\bar{x}_1, 0, \boxplus)$ and $\alpha' = (\bar{x}_1, 1, \boxminus)$
 2: Initialize $\mathcal{V} = \{\alpha, \alpha'\}$ and $\mathcal{A} = \{(\mathsf{r}, \alpha), (\mathsf{r}, \alpha')\}$
 3: for $1 < i \leq K$ do
 4:    Let $\alpha = (\bar{x}_i, 0, \boxplus)$, $\alpha' = (\bar{x}_i, 1, \boxminus)$, $\beta = (\bar{x}_{i-1}, 0, \boxplus)$
 5:    Let $\mathcal{V} \leftarrow \mathcal{V} \cup \{\alpha, \alpha'\}$ and $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\beta, \alpha), (\beta, \alpha')\}$
 6: end for
 7: BranchAndPrune($K + 1$, $(\bar{x}_K, 0, \boxplus)$)
 8: Let $X = \{x(\theta) \mid \theta \in \mathcal{V} \wedge |N^+(\theta)| = 0 \wedge \mu(\theta) = \boxplus\}$
 9: stop
10:
11: function BranchAndPrune($i$, $\beta$):
12: if $i > n \vee \mu = \boxminus$ then
13:    return
14: end if
15: Let $v = \rho^{-1}(i)$
16: Compute the equation $a_v^\top x = a_{v0}$ of the hyperplane through $x[U_v]$
17: Let $Z = \{z', z''\}$ be extensions of $x(\beta)$ to $v$, and $Z' = Z$
18: for $z \in Z$ do
19:    if $\exists \{u, v\} \in E \ \|x(\beta)_u - z\| \neq d_{uv}$ then
20:       Let $Z = Z \smallsetminus \{z\}$
21:    end if
22: end for
23: if $Z = \{z', z''\}$ then
24:    if $a_v^\top z' \leq a_{v0}$ then
25:       Let $\alpha = (z', 0, \boxplus)$, $\alpha' = (z'', 1, \boxplus)$
26:    else
27:       Let $\alpha = (z'', 0, \boxplus)$, $\alpha' = (z', 1, \boxplus)$
28:    end if
29: else if $Z = \{z\}$ then
30:    if $a_v^\top z \leq a_{v0}$ then
31:       Let $\alpha = (z, 0, \boxplus)$, $\alpha' = (Z' \smallsetminus \{z\}, 1, \boxminus)$
32:    else
33:       Let $\alpha = (z, 1, \boxplus)$, $\alpha' = (Z' \smallsetminus \{z\}, 0, \boxminus)$
34:    end if
35: else
36:    return
37: end if
38: Let $\mathcal{V} \leftarrow \mathcal{V} \cup \{\alpha, \alpha'\}$ and $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\beta, \alpha), (\beta, \alpha')\}$
39: for $\theta \in N^+(\beta)$ such that $\mu(\theta) = \boxplus$ do
40:    BranchAndPrune($i + 1$, $\theta$)
41: end for
42: return
```

$\Upsilon(y,v)$: *there are feasible leaf nodes* $\beta,\beta' \in \mathcal{V}_n$ *such that* $x(\beta) = y$, $\mathsf{p}(\beta) \cap \mathcal{V}_{\rho(v)-1} = \mathsf{p}(\beta') \cap \mathcal{V}_{\rho(v)-1}$ *and* $\mathsf{p}(\beta) \cap \mathcal{V}_{\rho(v)} \neq \mathsf{p}(\beta') \cap \mathcal{V}_{\rho(v)}$.

If $\Upsilon(y,v)$ holds, it is easy to show that $\mathsf{p}(\beta)\cap\mathcal{V}_{\rho(v)-1}$ contains a single feasible node with two feasible subnodes. With $\Upsilon(y,v)$ true, we let R^v be the Euclidean reflection operator with respect to the hyperplane through $y[U_v]$ (as discussed in p. 326). Define $\tilde{R}^v = I^{\rho(v)-1} \times (R^v)^{n-\rho(v)}$, i.e. $\tilde{R}^v y = (y_1,\dots,y_{i-1},R^v y_i,\dots,R^v y_n)$. This is a *partial reflection* of y which only acts on vertices past rank $i-1$.

We emphasize that for all $\ell \in \{i,\dots,n\}$ and for all $\alpha \in \mathcal{V}_\ell$ the set $\mathsf{p}(\alpha) \cap \mathcal{V}_i$ has a unique element, as it contains the unique node at level i on the path from α to the BP tree root node.

The following is a corollary to Lemma 3.3.

Corollary 4.1. *Let* $\alpha \in \mathcal{V}_{i-1}$ *for some* $i > 1$, $v = \rho^{-1}(i)$ *and* $N^+(\alpha) = \{\eta,\beta\}$ *with* $\mu(\eta) = \mu(\beta) = \boxplus$. *Then* $x(\eta)_v = R^v x(\beta)_v$.

Remark 4.2. If $\Upsilon(y,v)$ holds for some $y \in X$ and $v \in V \setminus V_0$, then by definition there are feasible leaf nodes in the BP tree, which implies that the considered DMDGP instance is YES.

An important consequence of Remark 4.2 is that all statements assuming $\Upsilon(y,v)$ and claiming a result with probability 1 implicitly also assume that the probability is conditional to the event of the DMDGP instance being a YES one. In particular, since the instance is YES, certain points *must* be placed at certain distances with probability 1, for otherwise the instance would be NO. This is evident in Prop. 4.4, Cor. 4.6, Cor. 4.7, and Thm. 4.9, where we state that certain real scalars and vectors must belong to certain finite sets with probability 1: the sense of these assertions, in this context, is that the Lebesgue measure of the set of YES instances not satisfying the result is zero in the set of all YES instances.

Lemma 4.3. *Let* $\alpha \in \mathcal{V}_{i-1}$ *for some* $i > 1$ *such that* $N^+(\alpha) = \{\eta',\beta'\}$, $u = \rho^{-1}(i)$; $v > u$ *with* $\rho(v) = \ell$, *and consider two feasible nodes* $\eta,\beta \in \mathcal{V}_\ell$ *such that* $\eta' = \mathsf{p}(\eta) \cap \mathcal{V}_i$ *and* $\beta' = \mathsf{p}(\beta) \cap \mathcal{V}_i$. *Then, with probability* 1, *the following statements are equivalent:*

(i) $\forall\, i \le j \le \ell$, $x(\beta'')_w = R^u x(\eta'')_w$, *where* $\eta'' = \mathsf{p}(\eta) \cap \mathcal{V}_j$, $\beta'' = \mathsf{p}(\beta) \cap \mathcal{V}_j$, *and* $w = \rho^{-1}(j)$;

(ii) $\forall\, i \le j \le \ell$, $\lambda(\eta'') = 1 - \lambda(\beta'')$, *with* $\eta'' = \mathsf{p}(\eta) \cap \mathcal{V}_j$ *and* $\beta'' = \mathsf{p}(\beta) \cap \mathcal{V}_j$.

Proposition 4.4. *Consider a subtree* $\mathcal{T}'$ *of* $\mathcal{T}$ *consisting of* $K+2$ *consecutive levels* $i-K-1,\dots,i$ *(where* $i \ge 2K+1$*), rooted at a single node* η *and such that all nodes at all levels are marked* $\boxplus$. *Let* $p = 2^{K+1}$ *and consider the set* $Y' = \{y_j \mid j \le p\}$ *of partial embeddings of* G *at the leaf nodes* $\{\alpha_j \mid j \le p\}$ *of* $\mathcal{T}'$. *Let* $u = \rho^{-1}(i-K-1)$ *and* $v = \rho^{-1}(i)$. *Then with probability 1 there are two distinct positive reals* r, r' *such that* $\|y_j(\alpha_j)_u - y_j(\alpha_j)_v\| \in \{r,r'\}$ *for all* $j \le p$.

Fig. 1 shows a graphical proof sketch of Prop. 4.4 for $K = 2$. Prop. 4.4 is useful in order to show that certain configurations of nodes within $\mathcal{T}$ can only occur with probability 0.

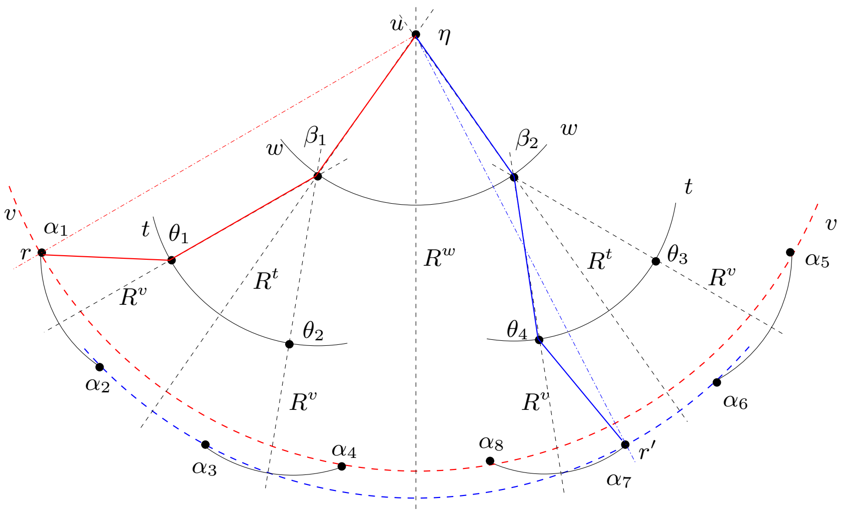


Fig. 1. Proof of Prop. 4.4 in $\mathbb{R}^2$. The arrangement of three segments gives rise, in general, to two distances r, r' between root and leaves.

Example 4.5. Consider a subtree $\mathcal{T}'$ of $\mathcal{T}$ like the one in Fig. 1 embedded in $\mathbb{R}^2$, and suppose that all nodes at level u, w, t are marked $\boxplus$, and further that only one node within α_1, α_2 is feasible, only one node within α_3, α_4 is feasible, only one node within α_7, α_8 is feasible, and α_5, α_6 are both infeasible. This must be due to a distance $d_{u'v}$ with $u' \leq u$. Consider now a circle C completely determined by its center at $y_1(\alpha_1)_{u'}$ and its radius $d_{u'v}$; if C also contains the points at the nodes $\alpha_1, \alpha_4, \alpha_8$ or the points at the nodes $\alpha_2, \alpha_3, \alpha_7$ then we must have $u' = u$, in which case also one of α_5, α_6 will be feasible (against the hypothesis). And the probability that C should contain the points at the nodes $\alpha_1, \alpha_3, \alpha_8$ or $\alpha_2, \alpha_4, \alpha_7$ is zero. Hence $\mathcal{T}'$ can only occur with probability 0. $\square$

We now exploit a generalization of Prop. 4.4 to build up towards the main result of this section, i.e. that partial reflections map valid embeddings to valid embeddings (Thm. 4.9).

Corollary 4.6. *Consider a subtree $\mathcal{T}'$ of $\mathcal{T}$ consisting of $K+q+1$ consecutive levels $i-K-q, \ldots, i$ (where $i \geq 2K+q$ and $q \geq 1$), rooted at a single node η and such that all nodes at all levels are marked $\boxplus$. Let $p = 2^{K+q}$ and consider the set $Y' = \{y_j \mid j \leq p\}$ of partial embeddings of G at the leaf nodes $\{\alpha_j \mid j \leq p\}$ of $\mathcal{T}'$. Let $u = \rho^{-1}(i-K-q)$ and $v = \rho^{-1}(i)$. Then with probability 1 there is a set $H^{uv} = \{r_j \mid j \leq 2^q\}$ of 2^q distinct positive reals such that $\|y_i(\alpha_i)_u - y_i(\alpha_i)_v\| \in H^{uv}$ for all $i \leq p$.*

The next corollary shows that distances spanning more than K vertices must all belong to certain finite sets of values for YES instances.

Corollary 4.7. *Let $y \in X$ and $v \in V \smallsetminus V_0$ such that $\Upsilon(y,v)$ holds. If $\{u,w\} \in E$ with $u < v < w$ and $\rho(w) - \rho(u) > K$ then $d_{uw} \in H^{uw}$ with probability 1.*

Corollary 4.8. *Let $y \in X$ and $v \in V \smallsetminus V_0$ such that $\Upsilon(y,v)$ holds. If $u \in V$ with $u > v$ then $R^v y_u$ belongs to a valid extension of $y[U_v]$.*

Finally, we state the main result of the section: if a DMDGP instance has a valid embedding y and v is a vertex where a "valid branching" (in the sense of the $\Upsilon(y,v)$ assumption) takes place in the BP algorithm, then the partial reflection of y with respect to v is also a valid embedding. We remark that the $\Upsilon(y,v)$ assumption only says that at v there is a BP search tree branching one of whose branch eventually leads to y, whilst the other ends up at any other valid embedding. Thm. 4.9 states that in this case the partial reflection of y w.r.t. v is also valid.

Theorem 4.9. *Let $y \in X$ and $v \in V \smallsetminus V_0$ such that $\Upsilon(y,v)$ holds. Then $\tilde{R}^v y \in X$ with probability 1.*

5 Symmetry and Number of Solutions

Our strategy for proving that YES instances of the DMDGP have power of two solutions with probability 1 is as follows. We map embeddings $y \in X$ to binary sequences $\chi \in \{0,1\}^n$ describing the "branching path" in the tree $\mathcal{T}$. We define a symmetry operation on χ by flipping its tail from a given component i (this operation is akin to branching at level i). We show that the cardinality of the group of all such symmetries is a power of two by bijection with a set of binary sequences. Finally we prove that the cardinality of the symmetry group is the same as $|X|$.

For all leaf nodes $\alpha \in \mathcal{V}$ with $\mu(\alpha) = \boxplus$ let $\chi(\alpha) = (\lambda(\beta) \mid \beta \in \mathsf{p}(\alpha))$; since embeddings in X are also in correspondence with leaf $\boxplus$-nodes of $\mathcal{T}$ by Alg. 1, Step 8, χ defines a relation on $X \times \{0,1\}^n$.

Lemma 5.1. *With probability 1, the relation χ is a function.*

Let $\Xi = \{\chi(y) \mid y \in X\}$. For $y \in X$ let y^i be its subsequence $(x_1, \ldots, x_i)$. We extend χ to be defined on all such subsequences by simply setting $\chi^i = (\chi(y)_1, \ldots, \chi(y)_i)$; $\chi(y)$ is valid if y is a valid embedding.

Let $N = \{1, \ldots, n\}$ and g be the $n \times n$ binary matrix such that $g_{ij} = 1$ if $i \le j$ and 0 otherwise (the upper triangular $n \times n$ all-1 matrix); let g_i be its i-th row vector and $\Gamma = \{g_i \mid i \in N\}$. Consider the elementwise modulo-2 addition in the set $\mathbb{F}_2^n$ (denoted $\oplus$): this endows $\mathbb{F}_2^n$ with an additive group structure with identity $e = (0, \ldots, 0)$ where each element is idempotent. Thus, $\mathcal{G} = (\mathbb{F}_2^n, \oplus) \cong C_2^n$. This group naturally acts on itself (and subsets thereof) using the same $\oplus$ operation. It is not difficult to prove that Γ is a set of group generators for $\mathcal{G}$ and a linearly independent set of the vector space $\mathcal{V}$ given by $\mathcal{G}$ with scalar multiplication over $\mathbb{F}_2$. For all $S \subseteq N$, let

$$g_S = \bigoplus_{i \in S} g_i,$$

and define a mapping $\phi : \mathcal{P}(N) \to \mathcal{G}$ given by $\phi(S) = g_S$.

Lemma 5.2. *ϕ is injective.*

The following result shows essentially that groups of partial reflections have power of two cardinality.

Lemma 5.3. *For all $H \subseteq \Gamma$, $|\langle H \rangle| = 2^{|H|}$.*

Let I be the set of levels of $\mathcal{T}$ for which from all nodes with two valid children there is a path going to a feasible leaf through both children. Let $L = \{g_i \in \Gamma \mid i \in I\}$ and $\Lambda = \langle L \rangle$ be the subgroup of $\mathcal{G}$ of "allowed partial reflections" generated by L. In the following (the main result of this section) we relate partial reflections to χ representations of valid embeddings. We show that any valid embedding, in its χ representation, generates the whole set of valid embeddings by means of the action of the group of allowed partial reflections.

Theorem 5.4. *If $\Xi \neq \emptyset$, for all $\xi \in \Xi$ we have $\xi \oplus \Lambda = \Xi$ with probability 1.*

The main result of the paper is now simply a corollary of Thm. 5.4.

Corollary 5.5. *If a DMDGP instance is YES, $|X|$ is a power of two with probability 1.*

6 Counterexamples

6.1 Disproving the "Power of Two" Conjecture

We first discuss a class of counterexamples to the conjecture that *all* DMDGP instances have a number of solutions which is a power of two (also see Lemma 5.1 in [22]). All these counterexamples are hand-crafted and have the property that two distinct embeddings x, x' have at least a level i where $x_i = x'_i$, which is an event which happens with probability 0. For any $K \geq 1$, let $n = K + 3$, $V = \{1, \ldots, n\}$, $E = \{\{i,j\} \mid 0 < i - j \leq K\} \cup \{\{1,n\}\}$ and $d_{ij} = 1$ for all $\{i,j\} \in E$. The first $n - 2 = K + 1$ points can be embedded in the vertices of a regular simplex in dimension K; then either $x_{n-1} = x_1$ or x_{n-1} is the symmetric position from x_1 with respect to the hyperplane through $\{x_2, \ldots, x_{n-2}\}$. In the first case, the two positions for x_n are valid, in the second only $x_n = x_2$ is possible (see Fig. 2 for the 2-dimensional case), yielding a YES instance where $|X| = 6$.

6.2 Necessity of Immediate Predecessors

Lastly, Fig. 3 shows an example where the *(ii)* $\Rightarrow$ *(i)* implication of Lemma 4.3 fails for instances in DDGP $\smallsetminus$ DMDGP. This shows that any generalization of

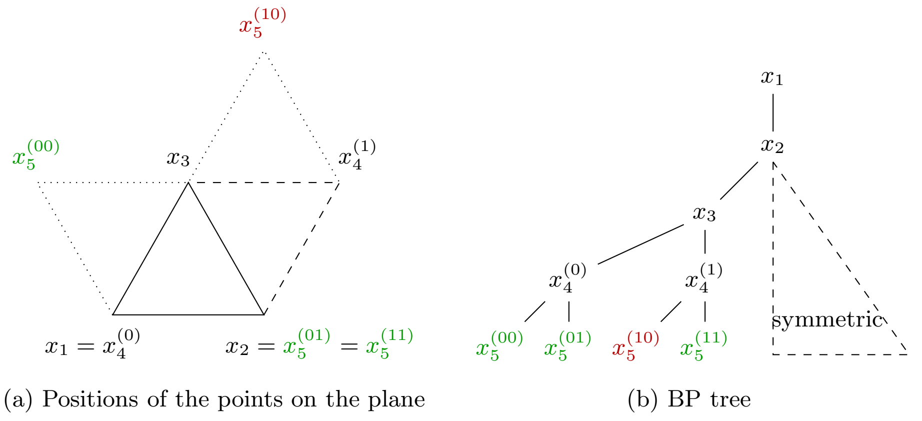

$x_5^{(10)}$
$x_5^{(00)}$
x_3
$x_4^{(1)}$
$x_1 = x_4^{(0)}$
$x_2 = x_5^{(01)} = x_5^{(11)}$
(a) Positions of the points on the plane
x_1
x_2
x_3
$x_4^{(0)}$
$x_4^{(1)}$
symmetric
$x_5^{(00)}$ $x_5^{(01)}$ $x_5^{(10)}$ $x_5^{(11)}$
(b) BP tree

Fig. 2. The counterexample in the case $K = 2$. Embeddings $x_5^{(00)}$, $x_5^{(01)}$, and $x_5^{(11)}$ are valid, while $x_5^{(10)}$ is not.

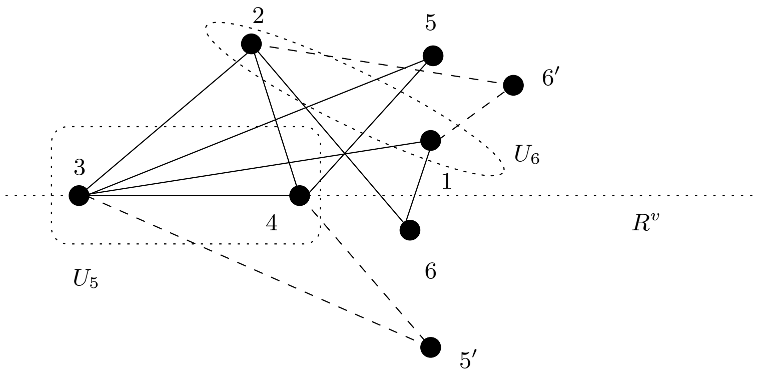


Fig. 3. A counterexample to Lemma 4.3 applied to DDGP ∖ DMDGP

our result to the DDGP is nontrivial. Let $V = \{1, \ldots, 6\}$ (the graph drawing is the same as the embedding in $\mathbb{R}^2$). The nodes $5', 6'$ linked with dashed lines show alternative node placements. Let $U_5 = \{3, 4\}$ and $U_6 = \{1, 2\}$. The line through the points $3, 4$ does not provide a valid reflection mapping 6 to $6'$. This happens because U_6 does not consist of the two *immediate* predecessors of 6.

7 Conclusion

In this paper we showed that YES instances of the DDGP have a number of solutions which is a power of two with probability 1. This settles a question which arose from an empirical observation in [22]. One of the partial results (Thm. 5.4) leading to the proof of this fact will also have practical implications, since all solutions can be expressed in function of one solution by means of a set of flip operations on binary sequences; we are going to test this idea computationally in future work.

Acknowledgments. The authors would like to thank the Brazilian research agencies FAPESP and CNPq, the French research agency CNRS and École Polytechnique, for financial support.

References

1. Lavor, C., Liberti, L., Maculan, N.: Computational experience with the molecular distance geometry problem. In: Pintér, J. (ed.) Global Optimization: Scientific and Engineering Case Studies, pp. 213–225. Springer, Berlin (2006)
2. Liberti, L., Lavor, C., Maculan, N., Marinelli, F.: Double variable neighbourhood search with smoothing for the molecular distance geometry problem. Journal of Global Optimization 43, 207–218 (2009)
3. Saxe, J.: Embeddability of weighted graphs in k-space is strongly NP-hard. In: Proceedings of 17th Allerton Conference in Communications, Control and Computing, pp. 480–489 (1979)
4. Huang, H.X., Liang, Z.A., Pardalos, P.: Some properties for the Euclidean distance matrix and positive semidefinite matrix completion problems. Journal of Global Optimization 25, 3–21 (2003)
5. Hendrickson, B.: The molecule problem: exploiting structure in global optimization. SIAM Journal on Optimization 5, 835–857 (1995)
6. Eren, T., Goldenberg, D., Whiteley, W., Yang, Y., Morse, A., Anderson, B., Belhumeur, P.: Rigidity, computation, and randomization in network localization. IEEE Infocom Proceedings, 2673–2684 (2004)
7. Krislock, N., Wolkowicz, H.: Explicit sensor network localization using semidefinite representations and facial reductions. SIAM Journal on Optimization 20, 2679–2708 (2010)
8. Gunther, H.: NMR Spectroscopy: Basic Principles, Concepts, and Applications in Chemistry. Wiley, New York (1995)
9. Schlick, T.: Molecular modelling and simulation: an interdisciplinary guide. Springer, New York (2002)
10. Santana, R., Larrañaga, P., Lozano, J.: Combining variable neighbourhood search and estimation of distribution algorithms in the protein side chain placement problem. Journal of Heuristics 14, 519–547 (2008)
11. Lavor, C., Mucherino, A., Liberti, L., Maculan, N.: Discrete approaches for solving molecular distance geometry problems using NMR data. International Journal of Computational Biosciences 1(1), 88–94 (2010)
12. Lavor, C., Liberti, L., Maculan, N., Mucherino, A.: The discretizable molecular distance geometry problem. Computational Optimization and Applications doi: 10.1007/s10589-011-9402-6
13. Liberti, L., Lavor, C., Maculan, N.: A branch-and-prune algorithm for the molecular distance geometry problem. International Transactions in Operational Research 15, 1–17 (2008)
14. Mucherino, A., Lavor, C., Liberti, L.: The discretizable distance geometry problem. To appear in Optimization Letters

15. Lavor, C., Lee, J., John, A.L.S., Liberti, L., Mucherino, A., Sviridenko, M.: Discretization orders for distance geometry problems. Optimization Letters doi: 10.1007/s11590-011-0302-6
16. Lavor, C., Mucherino, A., Liberti, L., Maculan, N.: On the computation of protein backbones by using artificial backbones of hydrogens. Journal of Global Optimization 50, 329–344 (2011)
17. Liberti, L., Lavor, C., Mucherino, A., Maculan, N.: Molecular distance geometry methods: from continuous to discrete. International Transactions in Operational Research 18, 33–51 (2010)
18. Lavor, C., Liberti, L., Maculan, N., Mucherino, A.: Recent advances on the discretizable molecular distance geometry problem. European Journal of Operational Research (accepted / invited survey)
19. Blumenthal, L.: Theory and Applications of Distance Geometry. Oxford University Press, Oxford (1953)
20. Connelly, R.: Generic global rigidity. Discrete Computational Geometry 33, 549–563 (2005)
21. Brady, T., Watt, C.: On products of Euclidean reflections. American Mathematical Monthly 113, 826–829 (2006)
22. Lavor, C., Liberti, L., Maculan, N.: The discretizable molecular distance geometry problem. Technical Report q-bio/0608012, arXiv (2006)
23. Dong, Q., Wu, Z.: A geometric build-up algorithm for solving the molecular distance geometry problem with sparse distance data. Journal of Global Optimization 26, 321–333 (2003)
24. Coope, I.: Reliable computation of the points of intersection of n spheres in $\mathbb{R}^n$. Australian and New Zealand Industrial and Applied Mathematics Journal 42, C461–C477 (2000)

A Appendix: Proofs

Lemma A.1 (3.1). *If $|N(v) \cap \gamma(v)| = K$ then there are at most two distinct extensions of x that are valid for G^v. If one valid extension exists, then with probability 1 there are exactly two distinct valid extensions.*

Proof. Since $|N(v) \cap \gamma(v)| = K$, $U_v = N(v) \cap \gamma(v)$ and v is at the intersection of exactly K spheres in $\mathbb{R}^K$ (each centered at x_u with radius d_{uv}, where $u \in U_v$). The position $z \in \mathbb{R}^K$ of v must then satisfy:

$$\forall u \in U_v \quad \|z - x_u\| = d_{uv} \Rightarrow \|z\|^2 - 2x_u \cdot z + \|x_u\|^2 = d_{uv}^2. \tag{6}$$

As in [23], we choose an arbitrary $w \in U_v$, say $w = \max_< U_v$, and subtract from the Eq. (6) indexed by w the other equations of (6), obtaining the system:

$$\left.\begin{array}{r} \forall u \in U_v \smallsetminus \{w\} \quad 2(x_u - x_w) \cdot z = (\|x_u\|^2 - d_{uv}^2) - (\|x_w\|^2 - d_{wv}^2) \\ \|z\|^2 - 2x_w \cdot z + \|x_w\|^2 = d_{wv}^2. \end{array}\right\} \tag{7}$$

The system (7) consists of a set of $K-1$ linear equations and a single quadratic equation in the K-vector z. We write the linear equations as the system $Az = b$, where the (u, j)-th component of A is $2(x_{uj} - x_{wj})$, the u-th component of b is $\|x_u\|^2 - \|x_w\|^2 - d_{uv}^2 + d_{wv}^2$, A is $(K-1) \times K$ and $b \in \mathbb{R}^{K-1}$. By strict simplex inequality, A has full rank (for otherwise $\sum_{u \neq w} \lambda_u (x_u - x_w) = 0$ implies that x_w is in the span of $\{x_u \mid u \in U_v\}$, and hence that $\Delta_{K-1}(U_v) = 0$); so without loss of generality assume that the square matrix B formed by the first $K-1$ columns of A is invertible. Let z_B be the vector consisting of the first $K-1$ components of z; then the linear part (first $K-1$ equations) of (7) yields $z_B = B^{-1}(b - N z_K)$, where $N = 2(x_{uK} - x_{wK} \mid u \in U_v \smallsetminus \{w\}) \in \mathbb{R}^{K-1}$. After replacement of z_B in (7) with $z_B(z_K)$, we obtain the following quadratic equation in z_K:

$$(\|\bar{N}\|^2 + 1) z_K^2 - 2((\bar{b} + x_{wB})\bar{N} + x_{wK}) z_k + (\|x_{wB} - \bar{b}\|^2 + x_{wK}^2 - d_{wv}^2) = 0, \tag{8}$$

where $\bar{b} = B^{-1} b$ and $\bar{N} = B^{-1} N$. If the discriminant of (8) is negative then no extension of $\bar{x}$ to v is possible and the result follows. If the discriminant is nonnegative, (8) has solutions z'_K, z''_K yielding points $z' = (z_B(z'_K), z'_K)$ and $z'' = (z_B(z''_K), z''_K) \in \mathbb{R}^K$, which are distinct with probability 1 because the discriminant is zero with probability 0. The extended embeddings, distinct with probability 1, are given by (x, z') and (x, z''). □

Lemma A.2 (3.2). *If $|N(v) \cap \gamma(v)| > K$ then, with probability* 1, *there is at most one extension of x.*

Proof. Consider a subset $S \subseteq N(v) \cap \gamma(v)$ such that $|S| = K + 1$ and $S \supseteq U_v$. Either there is at least one point x_v such that (x, x_v) is an embedding of $G[S \cup \{v\}]$ that is valid w.r.t. the system:

$$\forall u \in S \quad \sum_{k \le K} (x_{vk}^2 - 2x_{uk}x_{vk} + x_{uk}^2) = d_{uv}^2, \tag{9}$$

or the system has no solution. In the latter case, the result follows, so we assume now that there is a point x_v satisfying (9). Since the points x_u are known for all $u \in S$, (9) is a quadratic system with K variables and $K+1$ equations. As in the proof of Lemma 3.1, we derive an equivalent linear system from (9). Since d satisfies the strict simplex inequalities on U_v with probability 1 and $S \supseteq U_v$, by [24] $\{x_u \mid u \in S\}$ are not co-planar and the system has exactly one solution. $\square$

Lemma A.3 (3.3). *With the notation of Lemma 3.1, if $\bar{x}$ is a valid embedding for $G[U_v]$, then z'' is a reflection of z' with respect to the hyperplane through the K points of $\bar{x}$.*

Proof. Any sphere in $\mathbb{R}^K$ is symmetric with respect to any hyperplane through its center; so the intersection of up to K spheres in $\mathbb{R}^K$ is symmetric with respect to the hyperplane containing all the centers. $\square$

Lemma A.4 (3.4). *At termination of Alg. 1, X contains all valid embeddings of G extending $\bar{x}$.*

Proof. Z exists with probability 1 by Lemma 3.1. Every embedding in X is valid because of Steps 17 and 19-20. No other valid extension of $\bar{x}$ exists because of Lemmata 3.1-3.2. $\square$

Proposition A.5 (3.5). *With probability 1, there is no level $i \leq n$ having two distinct feasible nodes $\beta, \theta \in \mathcal{V}_i$ such that $|\{\alpha \in N^+(\beta) \mid \mu(\alpha) = \boxplus\}| = 1$ and $|\{\alpha \in N^+(\theta) \mid \mu(\alpha) = \boxplus\}| = 2$.*

Proof. We show that for all $i \leq n$ the event of having two distinct nodes $\beta, \theta \in \mathcal{V}_i$, with $\rho^{-1}(i) = v$, such that β has one feasible subnode and θ has two has probability 0. Consider $T_v = N(v) \cap \gamma(v)$: if $|T_v| = K$ then by Lemma 3.1 β should have exactly two feasible subnodes with probability 1; since it only has one, the event $|T_v| = K$ occurs with probability 0. Since $|T_v| \geq K$ by (4), the event $|T_v| > K$ occurs with probability 1. Thus by Lemma 3.2 there is at most one valid embedding extending the partial embedding at v, which means that the two feasible subnodes of θ represent the same embedding, an event that occurs with probability 0. $\square$

Lemma A.6 (4.3). *Let $\alpha \in \mathcal{V}_{i-1}$ for some $i > 1$ such that $N^+(\alpha) = \{\eta', \beta'\}$, $u = \rho^{-1}(i)$; $v > u$ with $\rho(v) = \ell$, and consider two feasible nodes $\eta, \beta \in \mathcal{V}_\ell$ such that $\eta' = \mathsf{p}(\eta) \cap \mathcal{V}_i$ and $\beta' = \mathsf{p}(\beta) \cap \mathcal{V}_i$. Then, with probability 1, the following statements are equivalent:*

(i) $\forall\, i \leq j \leq \ell$, $x(\beta'')_w = R^u x(\eta'')_w$, *where* $\eta'' = \mathsf{p}(\eta) \cap \mathcal{V}_j$, $\beta'' = \mathsf{p}(\beta) \cap \mathcal{V}_j$, *and* $w = \rho^{-1}(j)$;

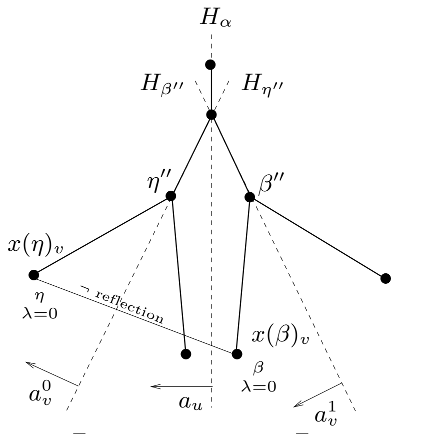
(ii) $\forall\, i \leq j \leq \ell$, $\lambda(\eta'') = 1 - \lambda(\beta'')$, *with* $\eta'' = \mathsf{p}(\eta) \cap \mathcal{V}_j$ *and* $\beta'' = \mathsf{p}(\beta) \cap \mathcal{V}_j$.

Proof. Let ${a_v^0}^\top x = a_{v0}^0$, ${a_v^1}^\top x = a_{v0}^1$ be the equations of the hyperplanes H_η, H_β defined respectively by $x(\eta)[U_v]$ and $x(\beta)[U_v]$, with the normals oriented as explained on page 326. We prove by induction on $\ell - i$ that the following assumption is equivalent to *(i)* and *(ii)*:

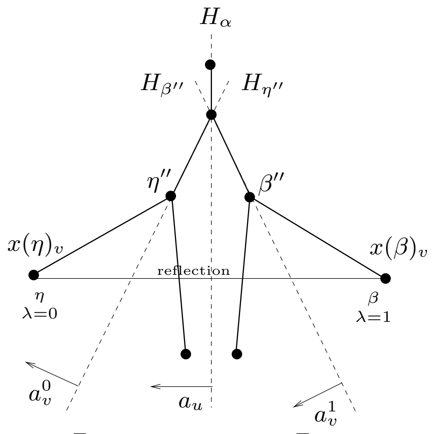
(*iii*) for all $i \le j \le \ell$, $x(\beta'')_w = R^u x(\eta'')_w$ and $a_u \cdot a_w^0 = a_u \cdot a_w^1$, where $\eta'' = \mathsf{p}(\eta) \cap \mathcal{V}_j$, $\beta'' = \mathsf{p}(\beta) \cap \mathcal{V}_j$, $w = \rho^{-1}(j)$, and a_w^0 and a_w^1 are the normal vectors of the hyperplanes $H_{\eta''}$ and $H_{\beta''}$ oriented as usual.

If $\ell = i$, then (*i*), (*ii*), and (*iii*) hold simultaneously. Indeed, $\eta = \eta'$ and $\beta = \beta'$, hence $x(\beta)_v = R^u x(\eta)_v$ (Lemma 3.3) and $\lambda(\eta) = 1 - \lambda(\beta)$ (Alg. 1, Steps 25 and 27). In addition, we have $H_\eta = R^u H_\beta$, therefore $|a_u \cdot a_v^0| = |a_u \cdot a_v^1|$. Because the orientation of a_v^0, a_v^1 is such that $a_u \cdot a_v^0, a_u \cdot a_v^1 \ge 0$, the result holds. Assume that the equivalence stated above holds for level $\ell - 1$, we show that it is still the case at level ℓ. In the sequel, denote $t = \rho^{-1}(\ell - 1)$.

(*i*) $\Leftrightarrow$ (*ii*). Suppose for all $i \le j < \ell$, $x(\beta'')_w = R^u x(\eta'')_w$ and $\lambda(\eta'') = 1 - \lambda(\beta'')$ (by the induction hypothesis, both statements are equivalent). Hence, $H_{\eta''} = R^u H_{\beta''}$ holds for all j, because the K points generating the hyperplanes either belong to H_α, or are reflections of each other. This is true in particular if we choose $\eta'', \beta'' \in \mathcal{V}_{\ell-1}$. In addition, if we use the induction hypothesis (*i*) $\Rightarrow$ (*iii*)), we have $a_u \cdot a_t^0 = a_u \cdot a_t^1$, so a_t^0, a_t^1 are directed similarly w.r.t a_u, and $\lambda(\eta) = 1 - \lambda(\beta)$ if and only if $x(\beta)_v = R^u x(\eta)_v$ (see Fig. 4).



(a) $a_v^{0\top} x(\eta)_v > a_{v0}^0$ and $a_v^{1\top} x(\beta)_v > a_{v0}^1$



(b) $a_v^{0\top} x(\eta)_v > a_{v0}^0$ and $a_v^{1\top} x(\beta)_v < a_{v0}^1$

Fig. 4. Proof of Lemma 4.3: Case (4a) shows the contradiction deriving from $\lambda(\eta) = \lambda(\beta) = 0$ (or $x(\beta)_v \ne R^u x(\eta)_v$), and case (4b) the situation that actually occurs

(*ii*) $\Rightarrow$ (*iii*). Suppose for all $i \le j \le \ell$, $\lambda(\eta'') = 1 - \lambda(\beta'')$. By the previous result, we also know that $i \le j \le \ell$, $x(\beta'')_w = R^u x(\eta'')_w$. It remains to prove that $a_u \cdot a_v^0 = a_u \cdot a_v^1$, i.e. that the angles θ_v^0 and θ_v^1 formed by these vectors have the same cosine. Notice once again that $H_\eta = R^u H_\beta$. By induction, we know that the angles θ_t^0, θ_t^1 formed by a_u and respectively a_t^0, a_t^1, have same cosine. With probability 1, the hyperplanes H_η, H_β are not parallel, hence their normal vectors cannot be identical, therefore, $\theta_t^0 = -\theta_t^1$ (see the illustration on Fig. 5). Denote θ^0, θ^1 the angles formed respectively by a_t^0 and a_v^0, and by a_t^1 and a_v^1. We also have, $H_{\eta''} = R^u H_{\beta''}$, where $\eta'', \beta'' \in \mathcal{V}_{\ell-1}$, hence the normal vectors of

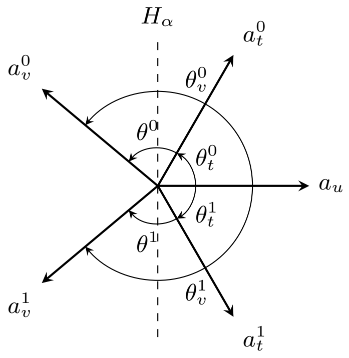


Fig. 5. Proof of Lemma 4.3: illustration of the fact that $a_u \cdot a_v^0 = a_u \cdot a_v^1$

these 4 hyperplanes are also symmetric, which implies $\theta^0 = -\theta^1$ or $\theta^0 = \pi - \theta^1$. By the definition of a_v^0 and a_v^1 (page 326), since the scalar products are positive, $-\pi/2 \leq \theta^0, \theta^1 \leq \pi/2$, thus $\theta^0 = -\theta^1$. Therefore, $\theta_v^0 = \theta_t^0 + \theta^0 = -\theta_t^1 - \theta^1 = -\theta_v^1$, which concludes this part of the proof. *(iii)* $\Rightarrow$ *(i)*. Obvious. □

Proposition A.7 (4.4). *Consider a subtree $\mathcal{T}'$ of $\mathcal{T}$ consisting of $K+2$ consecutive levels $i-K-1, \ldots, i$ (where $i \geq 2K+1$), rooted at a single node η and such that all nodes at all levels are marked $\boxplus$. Let $p = 2^{K+1}$ and consider the set $Y' = \{y_j \mid j \leq p\}$ of partial embeddings of G at the leaf nodes $\{\alpha_j \mid j \leq p\}$ of $\mathcal{T}'$. Let $u = \rho^{-1}(i-K-1)$ and $v = \rho^{-1}(i)$. Then with probability 1 there are two distinct positive reals r, r' such that $\|y_j(\alpha_j)_u - y_j(\alpha_j)_v\| \in \{r, r'\}$ for all $j \leq p$.*

Proof. Fig. 1 shows a graphical proof sketch for $K = 2$. With a slight abuse of notation, for a vertex $w \in V$ in this proof we denote by R^w the set of all reflections at level w. We order the α_j nodes so that the action of R^v on $(\alpha_1, \ldots, \alpha_p)$ is the permutation $\prod_{j \bmod 2=1}(j, j+1)$. Let $t = \rho^{-1}(i-1)$. Since all nodes are feasible, $\|y_j(\alpha_j)_v - y_j(\alpha_j)_t\| = d_{tv}$ and $\|y_j(\alpha_j)_u - y_j(\alpha_j)_t\| = d_{ut}$ for all $j \leq p$ (we remark that $\{t, v\}$ and $\{u, t\}$ must be in E by the definition of the DMDGP). With probability 1, the segments through $y_j(\alpha_j)_u$ and $y_j(\alpha_j)_t$ (where $j \leq p$) do not respectively lie within the hyperplanes defining the reflections R^v; and the same holds for the segments through $y_j(\alpha_j)_t$ and $y_j(\alpha_j)_v$. Thus, there is a set Q of positive reals $r_1, \ldots, r_p$ s.t. for all $j \leq p$ with $j \bmod 2 = 1$ we have $\|y_j(\alpha_j)_u - y_i(\alpha_j)_v\| = r_j$ and $\|y_{j+1}(\alpha_{j+1})_u - y_{j+1}(\alpha_{j+1})_v\| = r_{j+1}$, which shows $|Q| \leq p = 2^{K+1}$. By Lemma 4.3 the action of R^t on $(\alpha_1, \ldots, \alpha_p)$ is the permutation $\prod_{j \bmod 4=1}(j, j+3)(j+1, j+2)$: this implies that $r_j = r_{j+3}$ and $r_{j+1} = r_{j+2}$ for all $j \bmod 4 = 1$, which shows $|Q| \leq p/2 = 2^K$. Inductively, for a vertex w s.t. $i-K \leq \rho(w) \leq i-1$ the action of R^w is $\prod_{j \bmod 2^{j-\rho(w)+1}}(j, j + 2^{i-\rho(w)+1} - 1)(j+1, j+2^{i-\rho(w)+1} - 2) \cdots (j + 2^{i-\rho(w)} - 1, j + 2^{i-\rho(w)})$, which implies that $|Q| \leq 2^{K+1-i+\rho(w)}$. Therefore $\rho(w) = i - K$ proves that $|Q| \leq 2$. The case $|Q| = 1$ can only occur if $y_j(\alpha_j)_u$, $y_j(\alpha_j)_t$ and $y_j(\alpha_j)_v$ are collinear for all $j \leq p$, an event that occurs with probability 0. □

Corollary A.8 (4.6). *Consider a subtree $\mathcal{T}'$ of $\mathcal{T}$ consisting of $K+q+1$ consecutive levels $i-K-q,\ldots,i$ (where $i \geq 2K+q$ and $q \geq 1$), rooted at a single node η and such that all nodes at all levels are marked $\boxplus$. Let $p = 2^{K+q}$ and consider the set $Y' = \{y_j \mid j \leq p\}$ of partial embeddings of G at the leaf nodes $\{\alpha_j \mid j \leq p\}$ of $\mathcal{T}'$. Let $u = \rho^{-1}(i-K-q)$ and $v = \rho^{-1}(i)$. Then with probability 1 there is a set $H^{uv} = \{r_j \mid j \leq 2^q\}$ of 2^q distinct positive reals such that $\|y_i(\alpha_i)_u - y_i(\alpha_i)_v\| \in H^{uv}$ for all $i \leq p$.*

Proof. The proof of Prop. 4.4 can be generalized to span an arbitrary number of levels by induction on q. Two distances $r_{j_1}, r_{j_2} \in H^{uv}$ can only be equal by collinearity of some subsets of points, an event occurring with probability 0. $\square$

Corollary A.9 (4.7). *Let $y \in X$ and $v \in V \smallsetminus V_0$ such that $\Upsilon(y,v)$ holds. If $\{u,w\} \in E$ with $u < v < w$ and $\rho(w) - \rho(u) > K$ then $d_{uw} \in H^{uw}$ with probability 1.*

Proof. Since $\Upsilon(y,v)$ holds, then the DMDGP instance is YES and there must exist at least two feasible nodes at level $\rho(w)$ in $\mathcal{T}$. If $d_{uw} \notin H^{uw}$ the probability that a completely determined sphere contains two arbitrary points in $\mathbb{R}^K$ is zero. Since the instance is a YES one, however, the BP algorithm does not prune all feasible nodes due to d_{uw}. By Cor. 4.6 the only remaining possibility (which therefore occurs with probability 1) is that $d_{uw} \in H^{uw}$. $\square$

Corollary A.10 (4.8). *Let $y \in X$ and $v \in V \smallsetminus V_0$ such that $\Upsilon(y,v)$ holds. If $u \in V$ with $u > v$ then $R^v y_u$ belongs to a valid extension of $y[U_v]$.*

Proof. If there is no edge $\{w,u\} \in E$ with $\rho(u) - \rho(w) > K$ the result follows by Cor. 4.1. Otherwise, by Cor. 4.7, $d_{wu} \in H^{wu}$. As in the proof of Prop. 4.4, all pairs of points that are feasible w.r.t. d_{wu} are reflections of each other w.r.t. R^v. $\square$

Theorem A.11 (4.9). *Let $y \in X$ and $v \in V \smallsetminus V_0$ such that $\Upsilon(y,v)$ holds. Then $\tilde{R}^v y \in X$ with probability 1.*

Proof. We have to show that $\tilde{R}^v y$ is a valid embedding for $G = (V,E)$. Partition E into three subsets E_1, E_2, E_3, where $E_1 = \{\{t,u\} \in E \mid t,u < v\}$, $E_2 = \{\{t,u\} \in E \mid t,u \geq v\}$ and $E_3 = \{\{t,u\} \in E \mid t < v \wedge u \geq v\}$. For E_1, by definition $\|(\tilde{R}^v y)_t - (\tilde{R}^v y)_u\| = \|Iy_t - Iy_u\| = \|y_t - y_u\| = d_{tu}$ as claimed. For E_2, $\|(\tilde{R}^v y)_t - (\tilde{R}^v y)_u\| = \|R^v y_t - R^v y_u\| = \|y_t - y_u\| = d_{tu}$ because R^v is an isometry. For E_3, we aim to show that $\|Iy_t - R^v y_u\| = d_{tu}$. Since $y \in X$, by Lemma 3.4 there is a feasible leaf node α with $x(\alpha) = y$. Because $\Upsilon(y,v)$, $\exists \eta \in \mathcal{V}_{\rho(v)-1}$ such that $x(\eta) = y[\gamma(v)]$ and $N^+(\eta) = \{\beta, \beta'\}$ with $\mu(\beta) = \mu(\beta') = \boxplus$; we can assume without loss of generality that $\mathsf{p}(\alpha) \cap \mathcal{V}_{\rho(v)} = \{\beta\}$; furthermore, again by $\Upsilon(y,v)$, there is at least one feasible leaf node α' such that $\mathsf{p}(\alpha') \cap \mathcal{V}_{\rho(v)} = \{\beta'\}$. Let $\{\omega\} = \mathsf{p}(\alpha) \cap \mathcal{V}_{\rho(u)}$ and $\{\omega'\} = \mathsf{p}(\alpha') \cap \mathcal{V}_{\rho(u)}$. Because ω' is feasible, $\|x(\omega')_t - x(\omega')_u\| = d_{tu}$; because η is an ancestor of both α and α' at level $\rho(v)-1$ and $t < v$, $\mathsf{p}(\alpha') \cap \mathcal{V}_{\rho(t)} = \mathsf{p}(\alpha) \cap \mathcal{V}_{\rho(t)}$, which implies that $x(\omega')_t = x(\omega)_t = y_t$. Thus, $\|y_t - y_u\| = d_{tu} = \|y_t - x(\omega')_u\|$. Furthermore, because $\beta' \in \mathsf{p}(\omega') \cap \mathcal{V}_{\rho(v)}$,

$x(\omega')$ extends $x(\beta')$. By Alg. 1, Steps 25 and 27, $\lambda(\beta) = 1 - \lambda(\beta')$. Because α is feasible, at every level $\rho(u') \in V$ such that $v \le u' < u$ the node $\theta \in \mathsf{p}(\alpha) \cap \mathcal{V}_{\rho(u')}$ has $f \in \{1, 2\}$ feasible subnodes; by Prop. 3.5, the node $\theta' \in \mathsf{p}(\alpha') \cap \mathcal{V}_{\rho(u')}$ also has f feasible subnodes. If $f = 2$, by Cor. 4.8 it is possible to choose α' so that $\lambda(\theta') = 1 - \lambda(\theta)$ with probability 1; if $f = 1$ then by Alg. 1, Steps 31 and 33, all feasible nodes inherit the same λ value as their parents, so $\lambda(\theta') = 1 - \lambda(\theta)$. By Lemma 4.3, $x(\omega')_u = R^v y_u$ with probability 1. Hence $\|y_t - R^v y_u\| = d_{tu}$ as claimed. □

Lemma A.12 (5.1). *With probability 1, the relation χ is a function.*

Proof. For χ to fail to be well-defined, there must exist an embedding x which is in relation with two distinct binary sequences χ', χ'', which corresponds to the discriminant of the quadratic equation in the proof of Lemma 3.1 taking value zero at some rank $> K$, which happens with probability 0. □

Lemma A.13 (5.2). *ϕ is injective.*

Proof. We show that for all $S, T \subseteq N$, if $g_S = g_T$ then $S = T$.

$$\begin{array}{lll}
 & & g_S = g_T \\
 & \Rightarrow & \bigoplus_{i\in S} g_i = \bigoplus_{i\in T} g_i \\
 & \Rightarrow & \bigoplus_{i\in S} g_i \oplus \bigoplus_{i\in T} g_i^{-1} = e \\
\text{idempotency} & \Rightarrow & \bigoplus_{i\in S} g_i \oplus \bigoplus_{i\in T} g_i = e \\
g_i \oplus g_i = g_i^2 & \Rightarrow & \bigoplus_{i\in S\triangle T} g_i \oplus \bigoplus_{i\in S\cap T} g_i^2 = e \\
\text{idempotency} & \Rightarrow & \bigoplus_{i\in S\triangle T} g_i = e \\
\text{linear independence} & \Rightarrow & S\triangle T = \emptyset \\
 & \Rightarrow & S = T.
\end{array}$$

This concludes the proof. □

Lemma A.14 (5.3). *For all $H \subseteq \Gamma$, $|\langle H\rangle| = 2^{|H|}$.*

Proof. The restriction of function ϕ to $\mathcal{P}(H)$ is injective by Lemma 5.2. Furthermore, each element g of $\langle H\rangle$ can be written as $\bigoplus_{i\in S} g_i$ for some $S \subseteq H$ because H is a spanning set for the vector space H over $\mathbb{F}_2^n$, which is setwise equal to the group $\langle H\rangle$. Thus ϕ is surjective too. Hence ϕ is a bijection between $\mathcal{P}(H)$ and $\langle H\rangle$, which yields the result. □

Theorem A.15 (5.4). *If $\Xi \neq \emptyset$, for all $\xi \in \Xi$ we have $\xi \oplus \Lambda = \Xi$ with probability 1.*

Proof. ($\Rightarrow$) We show that $\xi \oplus \Lambda \subseteq \Xi$ with probability 1; because $\langle L\rangle = \Lambda$ it suffices to show that $\xi \oplus g_i \in \Xi$ for an arbitrary $g_i \in L$, i.e. that there exists a valid embedding $w \in X$ such that $\chi(w) = \xi \oplus g_i$. Let $y \in \chi^{-1}(\xi)$ and $v = \rho^{-1}(i)$ such that $\Upsilon(y, v)$, and define $w = \tilde{R}^v y$ (where $\tilde{R}^v$ is defined in Thm. 4.9 above); by Thm. 4.9, $w \in X$. Let α' be the leaf node of $\mathcal{T}$ such that $x(\alpha') = y$; by Lemma 3.4, there is a leaf node β' such that $x(\beta') = w$. We have to show that for all $\ell \geq i$ the node $\beta \in \mathsf{p}(\beta') \cap \mathcal{V}_\ell$ is such that $\lambda(\beta) = 1 - \lambda(\alpha)$, where α is the node in $\mathsf{p}(\alpha') \cap \mathcal{V}_\ell$. We proceed by induction on ℓ. For $\ell = i$ this holds by Lemma 3.3. For $\ell > i$, the induction hypothesis allows us to apply Lemma 4.3 and conclude that the event $\lambda(\alpha) = 1 - \lambda(\beta)$ occurs with probability 1.

($\Leftarrow$) Now we show that $\Xi \subseteq \xi \oplus \Lambda$ with probability 1, i.e. for any $\eta \in \Xi$ there is $g \in \Lambda$ with $\xi \oplus g = \eta$. We proceed by induction on n, which starts when $n = K+1$: if $K+1 \not\in I$ then $|\Xi| = 1$, $L = \emptyset$ and the theorem holds; if $K+1 \in I$ then $|\Xi| = 2$, $L = \{g_{K+1}\}$ and the theorem holds. Now let $n > K+1$; for all $j \in \{K+1, \ldots, n-1\}$ define $\Xi^j = \{\xi^j \mid \xi \in \Xi\}$ and $L^j = \{g_\ell \in \Gamma \mid \ell \in I \wedge \ell \leq j\}$. By the induction hypothesis, for all $\xi' \in \Xi^j$ $(\xi' \oplus \langle L^j \rangle = \Xi^j)$. Now, either $n \not\in I$ or $n \in I$; by Prop. 3.5, with probability 1 if $n \not\in I$ then nodes in $\mathcal{V}_{n-1}$ can only have zero or one feasible subnode (let B_1^n be the set of all such feasible subnodes), and if $n \in I$ then nodes in $\mathcal{V}_{n-1}$ can only have zero or two feasible subnodes β (let B_2^n be the set of all such feasible subnodes). In the former case we let $\Xi^n = \{\xi(x(\beta)) \mid \beta \in B_1^n\}$ and $L^n = L^{n-1}$; in the latter we let $\Xi^n = \{\xi(x(\beta)) \mid \beta \in B_2^n\}$ and $L^n = L^{n-1} \cup \{g_n\}$. In both cases it is easy to verify that the theorem holds for Ξ^n, L^n: in the former case it follows by the induction hypothesis, and in the latter case it follows because $g_n = (0, \ldots, 0, 1)$, namely, if $\eta \in \Xi$ and $n \in I$ then take $\xi = \eta \oplus g_n$ (the result follows by idempotency of g_n). □

Corollary A.16 (5.5). *If a DMDGP instance is feasible, $|X|$ is a power of two with probability 1.*

Proof. By Lemma 5.1 χ is a function with probability 1. Let $x, x' \in X$ be distinct; then by Alg. 1, Steps 25, 27, 31, and 33, the map $\chi : X \to \Xi$ is injective. By definition of Ξ it is also surjective, hence $|X| = |\Xi|$. By Thm. 5.4 $|\Xi| = |\chi \oplus \Lambda|$ for all $\chi \in \Xi$ with probability 1. It is easy to show that $|\chi \oplus \Lambda| = |\Lambda|$, so by Lemma 5.3 $|X|$ is a power of two with probability 1. □

Integration of an LP Solver into Interval Constraint Propagation*

Ernst Althaus[1,2], Bernd Becker[3], Daniel Dumitriu[1], and Stefan Kupferschmid[3]

[1] Johannes Gutenberg University, Mainz, Germany
{ernst.althaus,dumitriu}@uni-mainz.de
[2] Max Planck Institute for Computer Science, Saarbrücken, Germany
[3] Albert Ludwig University, Freiburg, Germany
{becker,skupfers}@informatik.uni-freiburg.de

Abstract. This paper describes the integration of an LP solver into iSAT, a Satisfiability Modulo Theories solver that can solve Boolean combinations of linear and nonlinear constraints. iSAT is a tight integration of the well-known DPLL algorithm and interval constraint propagation allowing it to reason about linear and nonlinear constraints. As interval arithmetic is known to be less efficient on solving linear programs, we will demonstrate how the integration of an LP solver can improve the overall solving performance of iSAT.

1 Introduction

We are considering the sat modulo theory (SMT) problem, which reads as follows. We are given a set of variables $\{x_1, \dots x_m\}$ and a set $C = \{c_1, \dots, c_n\}$ of constraints of some class of constraints over these variables, e.g. linear inequalities. Furthermore, we are given a Boolean structure over C, i.e. a Boolean formula ϕ, whose variables are the elements of C. The task is to decide the feasibility of ϕ, i.e. a Boolean assignment $b : C \mapsto \{\text{true}, \text{false}\}$ to the constraints, such that ϕ is satisfied for b and such that there is $x \in \mathbb{R}^m$ satisfying the constraints $\{c_i \mid b(c_i) = \text{true}\}$ and $\{\neg c_i \mid b(c_i) = \text{false}\}$. Hence we assume that the class of constraints is closed under negation.

SMT is an extension of the classical satisfiability problem. Such formulae arise in many applications dealing with the verification of *Hybrid Systems* [12].

The problem can be decided, if the feasibility problem of a conjunction of constraints of the class of constraints can be solved. The most prominent algorithm is the DPLL algorithm [6] in which the Boolean values for the constraints are determined via a search with backtracking, like in ordinary SAT algorithms. Furthermore, for each partial assignment of Boolean values, the corresponding set of constraints is checked for feasibility. There are many tricks to improve

* This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, www.avacs.org) and as part of its priority program "SPP 1307: Algorithm Engineering", grant AL 1139/1-2.

the running time, like the so-called *unit propagation* – if all but one literal of a clause are assigned to false, the remaining literal has to be true. In particular for the class of linear inequalities, there are many implementations to tackle the problem; we mention here only Yices [8], MathSAT [4], and OpenSMT [5].

We are interested in more complicated classes of constraints, i.e. we allow for arbitrary equalities and inequalities (strict and non-strict) composed from addition, multiplication, exponentiation, and (co-)sine. Hence, the problem under consideration becomes undecidable in general[1].

This large class of constraints allows us to model Hybrid Systems in a more precise way as lots of physical systems contain a quadratic, or exponential behavior that we want to analyze.

As we are dealing with transcendental functions the problem is in general undecidable, therefore we do not aim to solve it, but we either report the infeasibility or report a point $x \in \mathbb{R}^m$ such that all equalities and inequalities are satisfied up to some predifined accuracy in the arithmetic. In particular, strict inequalities are sometimes only satisfied non-strict. In order to do so, we have to assume that all variables x_i and all subformulae[2] have a bounded domain. We do so by an extension of the DPLL procedure in which we additionally allow the splitting of the domain of a variable. To become efficient, we use *interval constraint propagation* (ICP), i.e. given a constraint c_i with fixed assignment $b(c_i)$ and the current domains of the variables, we try to narrow the domains without removing any point satisfying the constraint. For example, if $x + y \leq 0$ is a constraint and $x \in [0, 1], y \in [-2, 2]$ are the current domains, we can narrow the domain of y to $[-2, 0]$ as there is no point in the original domain that satisfies $x + y \leq 0$ with $y > 0$. We refer to [2] for details.

This algorithm does not require a specialized solver for a particular class of constraints as long as the narrowing of the domains can be done efficiently. Nevertheless, the work of Gao et al. [10] indicates that an additional check for feasibility for all linear constraints can improve upon the running time. They employ an LP solver to characterize the feasible region described by the linear constraints and try to hand over this information to an ICP solver (they use the predecessor of our solver). Our work differs in that the overall search process is guided by the ICP solver and not by an LP solver as implemented in [10]. As ICP is known to be less efficient when solving linear systems we perform additional LP solver calls to detect unsatisfiable search parts earlier, and thus prevent the solver from making unnecessary ICP calls. Furthermore, our results demonstrate that the combination of an ICP solver and an LP solver increases the number of problems that we can solve, especially on unsatisfiable problem instances.

We call a specialized LP solver once the domain propagation reached a fixed point. The solver first tries to decide the feasibility or infeasibility using a solution/Farkas proof of a previous LP; if this is not successful, there are some

[1] It is well known that nonlinear arithmetic over the real numbers involving transcendental functions like *sin* is undecidable [19].

[2] We do not give a formal definition here and refer to [9] for details.

heuristics to decide whether the LP is solved or not. Once an LP has been shown to be infeasible, a small subset of the constraints causing the infeasibility based on the Farkas proof is given to the solver to speed up the search. This technique is called *conflict learning*.

The typical application of our algorithm is for model checking, in which the infeasibility of the formula means that the system is safe. If we report infeasible, this has to be safe and not corrupted due to rounding errors. Hence the domain narrowing is done in a conservative way, i.e. we always adopt the rounding mode such that the domain overapproximates the real domain. Using an LP solver, we have to make sure that we do not declare a feasible LP to be infeasible or report an infeasible subsystem that is feasible. In previous approaches, this was guaranteed using an LP solver based on rational arithmetic. In this paper, we show how a state-of-the-art floating point based LP solver can be used. We compare an approach that goes along the lines proposed by Dhiflaoui et al. [7] to an approaches that goes along the lines proposed by Neumaier and Shcherbina [15]. The later extension of this approach by Althaus and Dumitriu [1] is not required as all linear systems are bounded.

Notice that in this context, some linear systems are infeasible due to the strictness of some bounds; as a result, an arbitrary small change of the bounds can make an infeasible system feasible, hence rounding errors are very critical.

The paper is organized as follows. We first review the interval constraint programming approach of our underlying solver and then describe how we integrate an LP solver in Section 2. In Section 3, we show how a floating point based LP solver can be used to solve the linear programs including strict inequalities, thereby certifying the correctness of its result. Before giving a conclusion, we report on some experiments in Section 4.

2 Integration of an LP Solver into iSAT

In this section we present our approach that combines iSAT, a DPLL based interval constraint solver, and an LP solver. In order to do this we first provide a short introduction to iSAT (for a more detailed account please refer to [9]).

2.1 Introducing iSAT

In the following let φ be a Boolean combination of linear and nonlinear constraint formula. The front-end of iSAT computes normalized constraints and the *Conjunctive Normal Form* (CNF). After that we end up with a formula having the following syntax:

$$\begin{aligned}
\mathit{formula} &::= \{\mathit{clause} \wedge\}^* \mathit{clause} \\
\mathit{clause} &::= (\{\mathit{atom} \vee\}^* \mathit{atom}) \\
\mathit{atom} &::= \mathit{simple\_bound} \mid \mathit{arithmetic\_predicate} \\
\mathit{simple\_bound} &::= \mathit{variable} \sim \mathit{rational\_const} \\
\mathit{arithmetic\_predicate} &::= \mathit{variable} \sim \mathit{uop}\ \mathit{variable} \mid \\
&\quad\ \ \mathit{variable} \sim \mathit{variable}\ \mathit{bop}\ \mathit{variable} \\
&\quad\ \ \mathit{variable} \sim \mathit{variable}\ \mathit{bop}\ \mathit{rational\_const}
\end{aligned}$$

In the above syntax, *uop* and *bop* are unary and binary operation symbols respectively, including $+$, $-$, $\times$, $\sin(\cdot)$, etc., *rational_const* ranges over the rational constants, and $\sim \in \{<, \leq, =, \geq, >\}$. To illustrate this phase, consider the following formula:

$$(x \geq 0) \wedge (x \leq 10) \wedge ((\sin(1/3x) + \sqrt{x} \geq y) \implies (y \geq 1/4x + 3)) \tag{1}$$

First we eliminate the Boolean operator $\Rightarrow$ by applying a Tseitin transformation [20]. To do so, the implication will be replaced by a new auxiliary Boolean variable (b). The remaining formula is then normalized by introducing additional *real* variables r_1, r_2 and r_3 and the following constraints $r_1 = 1/3x$, $r_2 = \sin(r_1)$ and $r_3 = \sqrt{x}$. Sometimes we call the additional introduced variables (r_1, r_2, r_3) *auxiliary variables*. Finally, the normalized CNF problem looks as follows:

$$\begin{aligned}
&(x \geq 0) \wedge (x \leq 10) \wedge (\overline{b} \vee r_2 + r_3 < y \vee y \geq 3 + r_4) \wedge \\
&(r_2 + r_3 \geq y \vee b) \wedge (y \geq 3 + r_4 \vee b) \wedge \\
&(r_1 = 1/3x) \wedge (r_2 = \sin(r_1)) \wedge (r_3 = \sqrt{x}) \wedge (r_4 = 1/4x)
\end{aligned} \tag{2}$$

All clauses now have the syntax described above and can be transferred to the solver. Before describing the solving process in detail, we informally define the underlying semantics. A constraint formula φ is satisfied by a valuation of its variables if all its clauses are satisfied, that is, if at least one atom is satisfied in any clause. An atom is satisfied wrt. the standard interpretation of the arithmetic operators and the ordering relations over the reals. A constraint formula φ is *satisfiable* if there exists a satisfying valuation, referred to as a *solution* of φ. Otherwise, φ is *unsatisfiable*. We remark that by definition of satisfiability, a formula φ including or implying the empty clause, denoted by $\perp$, cannot be satisfied at all, i.e. if $\perp \in \varphi$ or φ is unsatisfiable.

Instead of real-valued variable valuations, iSAT manipulates interval ranges. By using the function $\rho : \mathit{Var} \rightarrow \mathbb{I}_{\mathbb{R}}$, where Var is a set of variables and $\mathbb{I}_{\mathbb{R}}$ is the set of interval ranges of $\mathbb{R}$, we define a range for each variable. Note, that we also support discrete variable domains (integer and Boolean). To this end, it suffices to clip the interval of integer variables accordingly, such that $[-3.4, 6.0)$ becomes $[-3, 5]$, for example. The Boolean domain is represented by $\mathbb{B} = [0, 1] \subset \mathbb{Z}$. If both ρ' and ρ are interval valuations, then ρ' is called a *refinement* of ρ if $\rho'(v) \subseteq \rho(v)$ for each variable $v \in \mathit{Var}$. The lower and upper interval borders of an interval $\rho(x)$ for a variable x can be encoded as simple bounds. We denote the lower and upper interval border of the interval $\rho(x)$ by $\mathit{lower}(\rho(x))$ and $\mathit{upper}(\rho(x))$, respectively. E.g., for the interval $\rho(x) = (-4, 9]$ we have $\mathit{lower}(\rho(x)) = (x > -4)$ and $\mathit{upper}(\rho(x)) = (x \leq 9)$.

Let x and y be variables, ρ be an interval valuation, and $\circ$ be a binary operation. Then $\rho(x \circ y)$ denotes the *interval hull* of $\rho(x) \hat{\circ} \rho(y)$ (i.e. the smallest enclosing interval which is representable by machine arithmetic), where the operator $\hat{\circ}$ corresponds to $\circ$ but is canonically lifted to sets. This is done analogously for unary operators. In order to compute the interval hull $\rho(x \circ y)$ we are using ICP. This allows us to narrow intervals of the variables, i.e. given a formula

$x \circ y \sim z$ and interval ranges for x, y and z, we look for intervals $\rho(x)$, $\rho(y)$ and $\rho(z)$ that are as small as possible without deleting any possible solution. By doing so iSAT can prune away definitive non-solutions and thus reducing the search space. Suppose the constraint $y = x^2$ and the interval ranges $x \in [3, 7]$ and $y \in [-2, 25]$. Using ICP we can compute a new lower bound for variable y. As the quadratic function ($y = x^2$) is strictly monotonic increasing for $x \in [3, 7]$ we know that the current lower bound of y is computed using the lower bound of x. We derive $\mathit{lower}(\rho(y)) = 3^2 = 9$ (the interval of y is updated to $[9, 25]$).

We say that an atom a is *inconsistent* under an interval valuation ρ, referred to as $\rho \sharp a$, if no values in the intervals $\rho(x)$ of the variables x in a satisfy the atom a, i.e.

$$\begin{array}{lll} \neg\exists v \in \rho(x) & : v \sim c & \text{if } \ a = (x \sim c), \\ \neg\exists v \in \rho(x), \neg\exists v' \in \rho(\circ y) & : v \sim v' & \text{if } \ a = (x \sim \circ y), \\ \neg\exists v \in \rho(x), \neg\exists v' \in \rho(y \circ z) & : v \sim v' & \text{if } \ a = (x \sim y \circ z) \end{array}$$

where $\sim \in \{<, \leq, =, \geq, >\}$. Otherwise a is *consistent* under ρ. For our purpose we do not need the definition of interval satisfaction. It is sufficient to talk about atoms which are still consistent. We remark that proving the satisfiability of an iSAT formula is not trivial. For more details we refer to [9, Subsection 4.5]. In Algorithm 1, the pseudocode of iSAT is given by ignoring the code between line 7 and line 11. Before the main iSAT routine starts, it is assumed that all the unit clause information contained in the original formula has already been propagated, which can sometimes allow us to derive tighter bounds. Once this is ensured, Algorithm 1 begins by making a decision, and splitting the interval range of a variable, e.g. splits a variable's range in half (line 3). This decision will be propagated in line 4. If a conflict is detected (e.g. a clause evaluates to $false$ during propagation) it will be analyzed in line 5. The conflict analysis routine uses the implication graph of the solver to compute the reasons for the conflict. By doing so a conflict clause is learned, allowing iSAT to prune off unsatisfiable parts of the search space. iSAT terminates in either line 5 or 13 with either *unsat*, *sat*, or *unknown*. If iSAT has managed it to split every problem variable up to a so called *minimum splitting width* (msw) and no conflict is detected the main DPLL loop is terminated and a satisfiability check for every problem clause is fueled in line 13.

2.2 Integration of an LP Solver

The integration of an LP solver affects the *normalization* and the *solving* part of iSAT. In the normalization part iSAT detects every linear constraint that is contained in the input formula. To get a better picture we will give an example. In equation 1 of the previous subsection there are three linear constraints ($x \geq 0$, $x \leq 10$, and $y \geq 1/4x + 3$). Every linear constraint that does not have the syntax of a *simple_bound* will be given to the LP solver. In this case the only linear constraint that is not a simple bound is $y \geq 1/4x + 3$. Here the normalization routine would transform the linear constraint into $-1/4x + y - s = 0$. In other

```
 1 Data: CNF F
 2 Result: sat, unsat or unknown
   /* Main DPLL loop. DecideVar returns 0 once the msw for        */
   /* all variables is reached, and no further decisions          */
   /* are possible.                                               */
 3 while decideVar() do
       /* Propagates current decision and unit constraints.       */
 4     if propagateICP() = Conflict then
           /* Function tries to resolve the conflict by backtracking.  */
           /* If conflict is unresolvable, problem is unsatisfiable.   */
 5         if analyseBacktrack() = Unresolvable then return unsat;
 6     end
       /* ICP did not find a conflict. Try to find a conflict     */
       /* among the linear constraints using an LP solver         */
 7     else if checkLPFeasibility() = Infeasible then
 8         insertCert() if analyseBacktrack() = Unresolvable then
 9             return unsat;
10         end
11     end
12 end
   /* Final test: Are all constraints satisfied?                  */
13 if allClausesSat() then return sat; else return unknown;
```

Algorithm 1. DPLL + ICP + LP

words we introduce for every linear constraint a so-called slack variable s. This way we produce an initially feasible LP tableau that can be transmitted to the LP solver; hence the normalization part produces the input for the iSAT solver and for the LP solver. The input for iSAT looks like:

$$\begin{aligned}
&(x \geq 0) \wedge (x \leq 10) \wedge (\overline{b} \vee r_2 + r_3 < y \vee y \geq r_4 + 3) \wedge \\
&(r_2 + r_3 \geq y \vee b) \wedge (s \geq 3 \vee b) \wedge \\
&(r_1 = 1/3x) \wedge (r_2 = \sin(r_1)) \wedge (r_3 = \sqrt{x}) \wedge \\
&(r_4 = 1/4x) \wedge (s = y - r_4)
\end{aligned} \tag{3}$$

And the input for the LP solver is:

$$-1/4x + y - s = 0 \tag{4}$$

Of course, iSAT is able to solve the formula given in equation 3 without the help of the LP solver. But as interval arithmetic is known to be less efficient when solving linear programs our aim is to improve the effectiveness of the solver by integrating an LP solver.

To do this we modify the iSAT procedure by adding additional feasibility checks for the linear constraints under the current interval valuation (line 7). These checks are performed after no conflict has been detected by the propagation phase (line 4). If the LP solver reports infeasible, a clause containing the negations of those column bounds that are responsible for the infeasibility is inserted into iSAT's learned clause database (line 8) thus preventing iSAT of entering this certain part of the search space again. To keep the number of literals small we compute these bounds from the Farkas' Lemma that is explained in the next section.

We further implemented the option of either adding the linear constraints only to the LP solver and just the nonlinear constraints are added to iSAT or the linear constraints are added to both solvers.

3 Solving the Linear Programs and Computation of Small Infeasible Subsets

We start our description assuming real arithmetic of the computer and describe necessary changes due to the floating point arithmetic afterwards. Besson [3] proposes a similar approach to ours, in that he computes infeasible subsystems with floating point arithmetic and certifies a correct result by solving a system of linear equations with rational arithmetic. His approach to compute infeasible subsystems for a system of linear inequalities including strict ones seems to be more complicated than ours.

Given a system of linear inequalities with some strict bounds as outlined in the previous section, deciding the feasibility can be achieved simply by maximizing the minimal distance to one of the strict inequalities, i.e. by solving the following linear program:

$$\begin{array}{rr} \max & \delta \\ s.t. & Ax = 0 \\ & x + u^s\delta \leq u \\ & x - \ell^s\delta \geq \ell \\ & \delta \geq 0, \end{array}$$

where we define $\ell_i^s = 1$ if the lower bound ℓ_i on x_i is strict and $\ell_i^s = 0$ otherwise, $u_i^s = 1$ if the upper bound u_i on x_i is strict and $u_i^s = 0$ otherwise.

If the LP is infeasible or has an objective function value of 0, the system of linear inequalities is infeasible, otherwise the system is feasible. A small but infeasible set of inequalities can be obtained then either from the Farkas certificate or from the dual solution. As the linear equations are globally valid, we do not include them in the certificate, but only a set of bounds.

More precisely, if the LP is infeasible, it has no solution with $\delta = 0$. Basically this means that the system has no solution, even if all strict bounds would be non-strict. Hence, we can drop δ and we know that the simplified linear system $Ax = 0, \ell \leq x \leq u$ is infeasible. Hence its Farkas system (a linear system which is feasible if and only if the given system is infeasible) is feasible and reads as follows:

$$\left.\begin{array}{rl} p^T A + q^T - r^T = & 0 \\ q^T u - r^T \ell = & -1 \\ q, r \geq & 0 \end{array}\right\} \qquad \text{(I)}$$

Notice that if we drop all variables with value 0, the linear system (I) is still feasible and hence the original LP without the corresponding constraints is still infeasible. Hence, the certificate consists of all lower bounds ℓ_i for which the corresponding Farkas system variable $r_i > 0$ and all upper bounds u_i for which the corresponding $q_i > 0$. The Farkas certificate for the (non-simplified) LP can be obtained from all LP solvers after the original LP has been solved and can be directly used for the simplified LP.

If the LP is feasible with objective function value 0, its dual has a solution with objective function value 0, i.e. the following system of linear equations is feasible:

$$\left.\begin{aligned} q^T u - r^T \ell &= 0 \\ p^T A + q^T - r^T &= 0 \\ q^T u^s + r^T \ell^s &= 1 \\ q, r &\geq 0 \end{aligned}\right\} \qquad \text{(II)}$$

Furthermore, if the system is feasible, the LP has objective function value 0 or is infeasible. Again the certificate consists of all lower bounds ℓ_i for which $r_i > 0$ and all upper bounds u_i for which $q_i > 0$ and can be obtained from the LP solution.

Assume now that the LP is solved with a state-of-the-art floating point solver. We discuss three alternative methods to certify the infeasibility and the computed Farkas certificate; we implemented the latter two of them. In case any of these methods fails, it returns that the feasibility status of the LP is unknown.

The first method was proposed by Dhiflaoui et al. [7]. They take the LP basis and try to verify its correctness using rational arithmetic. Basically, the basis gives a subset of the variables such that the system (I), respectively (II), has a solution with all non-basic variables having value 0, and the system of equations reduced to the basic variables has a unique solution which is non-negative. Given the basis, we can solve a system of linear equations and check that the solution is non-negative, in order to certify that the system (I), respectively (II), has a solution. This is done using rational arithmetic. The experiments of Dhiflaoui et al. show that for some instances, the time to solve these systems of linear equations is much higher than the time to solve the LP. This observation has been confirmed by a more efficient implementation by Koch [14].

In the second method, we solve a smaller system of equations in order to become more efficient. More precisely, we select the variables with floating point value larger than some $\varepsilon > 0$ and solve the corresponding subsystem of equations. Notice that these variables are a subset of the basic variables; this subset is strict if the basis is degenerate, which is often the case in practical applications. In this case, we get a linear system which is overdetermined. In Section 4, we show that these systems are often significantly smaller and can be solved very efficiently.

The third method was proposed by Neumaier and Shcherbina [15] and can be applied as all variables have finite bounds. We discuss that case of the infeasible LP first. Given the floating point solution (p, q, r), they compute pA using interval arithmetic. From this, they compute intervals for q and r such that the system $p^T A + q^T - r^T = 0$ definitely has a solution by setting $q_i = [\max(0, -\mathit{upper}((pA)_i), \max(0, -\mathit{lower}((pA)_i)]$ and $r_i = [\max(0, \mathit{lower}((pA)_i), \max(0, \mathit{upper}((pA)_i)]$. Then we certify, again using interval arithmetic, that $q^T u - r^T \ell < 0$. Notice that it suffices that the value is strictly negative, as a solution of the linear system can then be obtained by scaling. The certificate consists of all upper bounds u_i such that the interval q_i contains a non-zero and all lower bounds ℓ_i such that the interval r_i contains a non-zero. Notice that in contrast to the two methods above, it is possible that the certificate contains both the lower and upper bound of a variable.

If the LP has objective function value 0, we have to certify this with interval arithmetic. This is only possible if p can be represented exactly by floating point numbers and if during computation of pA the intervals remain point intervals. Still, the certificates are often trivial enough such that this is achieved. Notice that we cannot expect to obtain a method that can handle non-point intervals, since a slight change of the bounds can change the objective function value and hence the feasibility status.

4 Experiments

We have implemented our approach into iSAT. To become efficient, we further try to detect implications among the linear constraints as presented in [16] before starting the search. By doing so the number of LP solver calls is decreased. Other methods, like the elimination of variables by exploiting equalities appearing as top-level conjuncts of the formula or the simplification of the Boolean part of the input formula, both suggested in [5], have not yet been implemented; they would probably speed up our algorithm considerably. As LP solver, we use SoPlex [18,21]; we do not use Gurobi [11] or CPLEX [13], as we plan to certify the feasibility of LPs using the approach of Althaus and Dumitriu [1] in a future version, where we will need access to the LU decomposition of the matrix.

We tested the performance of our implementation on the QF-LRA benchmarks (quantifier-free linear real arithmetic) from SMT-LIB [17], which contains SMT problems having only linear constraints. In order to be able to use them within our ICP solver, we artificially make all variables bounded between -10^5 and 10^5. These instances can be solved by specialized solvers, which have much better running times. Nevertheless, we have chosen these benchmarks because they come from a standard benchmark library.

The iSAT solver is tuned to prove the unsatisfiability of a formula and returns *unknown* whenever it finds a candidate solution. Hence, for feasible solutions it often returns *unknown* and thus we cannot compare running times for those instances. Therefore, we restrict to unsatisfiable instances.

All experiments are performed on a 2.3 GHz Quad-Core AMD Opteron machine with 4 GB of physical memory running Ubuntu Linux with kernel version 2.6.32. We compiled our program with `g++` 4.4.3 with the optimization flag `-O2`.

4.1 Comparison of Different LP Solving Techniques

We compared four different methods to solve the linear systems, all of them giving certified correct results, i.e. results that are not corrupted by possible errors in the floating point arithmetic. In addition to the three methods described in Section 3, we use an LP solver based on rational arithmetic. For this purpose, we use the LP solver Yices [8], one of the fastest SMT solvers available, which is specialized to handle strict inequalities.

Table 1. Benchmarks: the first line indicates the solver for the LP, where *ICP w/o LPS* means that no LP solver is used, i.e. all linear constraints are only handled with the ICP-approach, *Neumaier/S* that we use the approach proposed by Neumaier and Shcherbina to certify the infeasibility of the linear program and ICP is not used, *ICP+our* that we use our approach and the ICP-approach of iSAT, *rational LPS* that we use the rational LP solver Yices and ICP is not used, and *our* that we use only our approach and ICP is not used. For each version, we provide the size of the search tree (*nodes*), the running time in seconds (*time*) and the result (*rs*), where *U* means that the solver correctly returned unsatisfiable and *?* that it returned *unknown*. The running time of the fastest approach was additionally marked with boldface.

	ICP w/o LPS			Neumaier/S			ICP+our			rational LPS			our approach		
Benchmark	nodes	rs	time	nodes	rs	time	nodes	rs	time	nodes	rs	time	nodes	rs	time
10cl.m-i.b	80	?	0.69	27197	?	249.29	1198	?	146.61	timeout			timeout		
10cl.w-c-s.b	0	U	**0.03**	0	U	**0.03**	0	U	0.04	0	U	0.04	0	U	**0.03**
10cl.w-c-s.in	timeout			timeout			timeout			timeout			timeout		
2cl.m-i.b	110	?	0.14	1120	?	1.55	179	?	4.54	1429	U	2.21	873	U	**1.17**
2cl.m-i.in	memout			562	?	3.81	timeout			2810	U	141.69	3889	U	**51.4**
2cl.w-c-s.b	0	U	0.01	0	U	0.01	0	U	0.02	0	U	0.02	0	U	**0**
2cl.w-c-s.in	timeout			25	U	**0.13**	20	U	16.22	27	U	0.66	25	U	0.56
3cl.m-i.b	119	?	0.19	2952	?	5.23	260	?	8.65	2166	U	7.08	2153	U	**3.52**
3cl.m-i.in	memout			1963	?	28.92	memout			timeout			4962	?	93.46
3cl.w-c-s.b	0	U	**0.01**	0	U	**0.01**	0	U	0.02	0	U	0.03	0	U	0.02
3cl.w-c-s.in	timeout			150	?	0.69	131	U	62.46	159	U	5.17	150	U	**3.57**
4cl.m-i.b	133	?	0.24	2808	?	7.65	272	?	19.61	3369	U	15.79	4045	U	**10.01**
4cl.m-i.in	memout			4265	?	89.62	memout			timeout			timeout		
4cl.w-c-s.b	0	U	**0.01**	0	U	**0.01**	0	U	0.03	0	U	0.02	0	U	0.02
4cl.w-c-s.in	timeout			351	?	2.48	486	U	157.94	394	U	20.51	379	U	**10.00**
5cl.m-i.b	142	?	0.29	5411	?	18.33	387	?	31.41	5724	U	34.9	7391	U	**21.27**
5cl.m-i.in	memout			10126	?	299	memout			timeout			timeout		
5cl.w-c-s.b	0	U	**0.02**	0	?	0	0	U	**0.02**	0	U	**0.02**	0	?	0.01
5cl.w-c-s.in	timeout			954	?	10.68	timeout			1469	U	63.3	964	U	**25.61**
6cl.m-i.b	156	?	0.37	16776	?	70.72	269	?	8.44	7358	U	70.69	11952	U	**38.84**
6cl.m-i.in	memout			5704	?	244.22	timeout			timeout			timeout		
6cl.w-c-s.b	0	U	0.03	0	U	**0.01**	0	U	0.02	0	U	0.03	0	U	0.02
6cl.w-c-s.in	timeout			1189	?	15.54	timeout			2227	U	216.17	2172	U	**58.50**
7cl.m-i.b	170	?	0.52	10231	?	65.95	594	?	51.88	11767	U	151.75	15465	U	**78.39**
7cl.m-i.in	memout			timeout			timeout			timeout			timeout		
7cl.w-c-s.b	0	U	0.03	0	U	0.03	0	U	**0.02**	0	U	0.03	0	U	0.03
7cl.w-c-s.in	timeout			2132	?	42.59	timeout			timeout			8866	U	**277.91**
8cl.m-i.b	179	?	0.58	11877	?	84.4	679	?	100.35	15778	U	207.85	21113	U	**115.45**
8cl.m-i.in	memout			timeout			timeout			timeout			timeout		
8cl.w-c-s.b	0	U	0.02	0	U	0.02	0	U	**0.01**	0	U	0.03	0	U	0.03
8cl.w-c-s.in	timeout			2560	?	68.28	timeout			timeout			timeout		
9cl.m-i.b	193	?	0.69	18838	?	162.94	1193	?	119.28	30200	U	286.39	28586	U	**164.25**
9cl.w-c-s.b	0	U	0.01	0	U	0.02	0	U	0.02	0	U	0.02	0	U	**0.00**
9cl.w-c-s.in	timeout			7791	?	281.53	timeout			timeout			timeout		

Due to internals of iSAT, it terminates sometimes with *unknown* before the time limit. These instances cannot be considered as correctly solved.

In Table 1, we show for each method the number of nodes in the search tree and the total running time for a typical subset of the instances, i.e. the *clock_ synchro* instances. With our approach for certifying the correctness of the floating point LP solver, 197 out of the 300 infeasible instances are solved, whereas only 193 are solved with the second best approach, i.e. using a rational LP solver. The small difference in the number of solved problem instances is explained by the fact that both approaches are very similar. However, the main advantage of our approach is that it needs approximately half of the solving time especially when focusing on instances with larger running times. Using the approach of Neumaier

Table 2. For our approach, we give some additional details for the instances that are solved. Beside the size of the search tree (*nodes*) and the running time (*time*) which can already be found in Table 1, we give the average size of the basis of the LPs (*basis*), the average size of the infeasible subsystem (*ifs*), the time needed by the LP solver (t_{LPS}), and the time needed for the Gaussian elimination (t_{Gauss}). Furthermore, we give the total number of linear systems that are declared to be infeasible by the floating point LP solver (*#inf*) and the number of times our method to certify infeasibility is not successful (*#?*).

Benchmark	nodes	time	basis	ifs	t_{LPS}	t_{Gaus}	#inf	#?
10cl.w-case-s.b	0	0.03	31	6	0	0	1	0
2cl.m-i.b	873	1.17	131	7	0.52	0.19	65	0
2cl.m-i.in	3889	51.4	249	10.91	24.6	20.49	367	65
2cl.w-case-s.b	0	0	15	6	0	0	1	0
2cl.w-case-s.in	25	0.56	87	15.76	0.03	0.45	17	0
3cl.m-i.b	2153	3.52	195	7.34	2	0.27	99	0
3cl.m-i.in	4962	93.46	388	11.79	58.8	21.45	461	230
3cl.w-case-s.b	0	0.02	17	6	0	0	1	0
3cl.w-case-s.in	150	3.57	128	17.88	0.51	2.72	71	7
4cl.m-i.b	4045	10.01	267	6.81	5.03	0.21	135	0
4cl.w-case-s.b	0	0.02	19	6	0	0	1	0
4cl.w-case-s.in	379	10	173	19.49	2.72	6.36	141	13
5cl.m-i.b	7391	21.27	347	6.97	11.57	0.44	173	0
5cl.w-case-s.b	0	0.01	21	0	0	0	1	1
5cl.w-case-s.in	964	25.61	222	19.79	9.77	12.4	233	14
6cl.m-i.b	11952	38.84	435	6.86	21.9	0.37	217	0
6cl.w-case-s.b	0	0.02	23	6	0	0	1	0
6cl.w-case-s.in	2172	58.5	275	20.35	27.61	22.74	403	61
7cl.m-i.b	15465	78.39	531	7.04	42.77	0.7	264	0
7cl.w-case-s.b	0	0.03	25	6	0	0	1	0
7cl.w-case-s.in	8866	277.91	332	20.91	171.34	70.51	1048	276
8cl.m-i.b	21113	115.45	635	6.8	59.07	0.69	279	0
8cl.w-case-s.b	0	0.03	27	6	0	0	1	0
9cl.m-i.b	28586	164.25	747	7.26	86.45	0.86	327	0
9cl.w-case-s.b	0	0	29	6	0	0	1	0

and Shcherbina to certify the infeasibility of the LPs, we can solve 174 of the instances.

We implemented the option to additionally handle linear constraints with the ICP approach and show for this setting the same numbers. This option turns out to produce worse results, e.g. it solves only 146 instead of 193 of the instances. It is not surprising that the combination of ICP and LP solving is less efficient: in this case ICP is redundant and less accurate compared to an LP solver. Furthermore, deriving that two linear constraints cannot hold at the same time by applying ICP steps can lead to a huge amount of arithmetic computations, causing the slowdown of the overall search process.

4.2 Detailed Evaluation of Our Certifying Approach

In Table 2 we show the details of our approach for certifying the correctness of the floating point based LP solver. For a subclass of the infeasible instances, we show the average size of the basis, the average size of the infeasible subsystem, the running time to solve the linear program, the running time to certify its correctness, the number of times our certifying is not successful, the number of nodes in the search tree and the total running time.

If our approach is successful, it returns exactly the same infeasible subsystem as the approach of Dhiflaoui et al., but solves a much smaller system of equations with rational arithmetic. More precisely, we solve a system with as many variables as the size of the infeasible subsystem, whereas they solve a system in the size of the basis. We show the differences in the size of the solved systems in Table 2 and do not report on the approach of Dhiflaoui et al. in our comparison on different LP certifying methods.

4.3 Summary

The integration of a floating point based LP solver into ICP can greatly reduce the running time needed to solve the benchmark instances.

Furthermore, the size of the infeasible subsets and hence the size of the system of linear equations we have to solve with rational arithmetic is very small on average, as one can see in column *ifs* (compare to column *basis*). This is helpful in two aspects: it gives small conflict clauses that can reduce the search space, and the running time for the Gaussian elimination is not the bottleneck – as opposed to solving the equation system given by the LP basis as in the work of Dhiflaoui et al. [7].

We believe that we become even more efficient, if we implement some further tricks to speed up the solution process, like storing LP solutions, carefully deciding which LPs should be solved or implementing further preprocessing techniques.

5 Conclusion

We have presented a tight integration of an LP solver into interval constraint propagation and experimented with our preliminary implementation. Already in this preliminary scenario which offers multiple possibilities for further optimization, the benefit of this integration is obvious.

As future work, we want to improve our approach, for instance by finding new methods to propagate bounds based on the linear constraints, as well as appropriate linear relaxations of nonlinear constraints.

References

1. Althaus, E., Dumitriu, D.: Fast and accurate bounds on linear programs. In: Proc. 8th International Symposium on Experimental Algorithms (SEA 2009). LNCS, vol. 5526, pp. 40–50. Springer, Heidelberg (2009)
2. Benhamou, F., Granvilliers, L.: Continuous and Interval Constraints. In: Handbook of Constraint Programming. Foundations of Artificial Intelligence, ch.16, pp. 571–603 (2006)
3. Besson, F.: On using an inexact floating-point LP solver for deciding linear arithmetic in an SMT solver. In: 8th International Workshop on Satisfiability Modulo Theories (2010)
4. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MathSAT 4 SMT Solver. In: Gupta, A., Malik, S. (eds.) Computer Aided Verification, Springer, Heidelberg (2008)
5. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The openSMT solver. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 150–153. Springer, Heidelberg (2010)
6. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
7. Dhiflaoui, M., Funke, S., Kwappik, C., Mehlhorn, K., Seel, M., Schömer, E., Schulte, R., Weber, D.: Certifying and repairing solutions to large LPs. How good are LP-solvers? In: Symposium of Discrete Algorithms (SODA), pp. 255–256 (2003)
8. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
9. Fränzle, M., Herde, C., Ratschan, S., Schubert, T., Teige, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. JSAT Special Issue on Constraint Programming and SAT 1, 209–236 (2007)
10. Gao, S., Ganai, M., Ivancic, F., Gupta, A., Sankaranarayanan, S., Clarke, E.: Integrating ICP and LRA solvers for deciding nonlinear real arithmetic. In: FMCAD (2010)
11. Gurobi Optimization: Gurobi Optimizer, http://www.gurobi.com/
12. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: Proc. of the 27th Annual Symposium on Theory of Computing, ACM Press, New York (1995)
13. IBM: ILOG CPLEX Optimizer, http://www01.ibm.com/software/integration/optimization/cplex-optimizer/
14. Koch, T.: The final NETLIB-LP results. Oper. Res. Lett. 32(2), 138–142 (2004)
15. Neumaier, A., Shcherbina, O.: Safe bounds in linear and mixed-integer linear programming. Math. Program. 99(2), 283–296 (2004)
16. Pigorsch, F., Scholl, C.: Using implications for optimizing state set representations of linear hybrid systems. In: GI/ITG/GMM Workshop MBMV (2009)

17. SMT-LIB: The Satisfiability Modulo Theories Library, http://goedel.cs.uiowa.edu/smtlib/
18. SoPlex: The Sequential object-oriented simplex, http://soplex.zib.de/
19. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. Univ. of California Press, Berkeley (1951)
20. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic, Part 2, pp. 115–125 (1970)
21. Wunderling, R.: Paralleler und objektorientierter Simplex-Algorithmus. Ph.D. thesis, Technische Universität Berlin (1996), http://www.zib.de/Publications/abstracts/TR-96-09/

A Saturation Algorithm for Homogeneous Binomial Ideals

Deepanjan Kesh and Shashank K. Mehta

Indian Institute of Technology, Kanpur - 208016, India
{deepkesh,skmehta}@cse.iitk.ac.in

Abstract. Let $k[x_1, \ldots, x_n]$ be a polynomial ring in n variables, and let $I \subset k[x_1, \ldots, x_n]$ be a homogeneous binomial ideal. We describe a fast algorithm to compute the saturation, $I : (x_1 \cdots x_n)^\infty$. In the special case when I is a toric ideal, we present some preliminary results comparing our algorithm with *Project and Lift* by Hemmecke and Malkin.

1 Introduction

1.1 Problem Description

Let $k[x_1, \ldots, x_n]$ be a polynomial ring in n variables over the field k, and let $I \subset k[x_1, \ldots, x_n]$ be an ideal. Ideals are said to be *homogeneous*, if they have a basis consisting of homogeneous polynomials. *Binomials* in this ring are defined as polynomials with at most two terms [5]. Thus, a binomial is a polynomial of the form $c\mathbf{x}^\alpha + d\mathbf{x}^\beta$, where c, d are arbitrary coefficients. *Pure difference binomials* are special cases of binomials of the form $\mathbf{x}^\alpha - \mathbf{x}^\beta$. Ideals with a binomial basis are called *binomial ideals*. Toric ideals, the kernel of a specific kind of polynomial ring homomorphisms, are examples of pure difference binomial ideals.

Saturation of an ideal, I, by a polynomial f, denoted by $I : f$, is defined as the ideal

$$I : f = \langle \{ \ g \in k[x_1, \ldots, x_n] \ : \ f \cdot g \in I \ \} \rangle.$$

Similarly, $I : f^\infty$ is defined as

$$I : f^\infty = \langle \{ \ g \in k[x_1, \ldots, x_n] \ : \ \exists a \in \mathbb{N}, f^a \cdot g \in I \ \} \rangle.$$

We describe a fast algorithm to compute the saturation, $I : (x_1 \cdots x_n)^\infty$, of a homogeneous binomial ideal I.

This problem finds applications in computing the radicals, minimal primes, cellular decompositions, etc., of a homogeneous binomial ideal, see [5]. This is also the key step in the computation of a toric ideal.

1.2 Related Work in Literature

The authors are not aware of any published work addressing this problem but there are several related works in the literature.

There are algorithms to compute the saturation of any ideal in $k[x_1, \ldots, x_n]$ (not just binomial ideals). One such algorithm is described in exercise 4.4.7 in [4]. It involves a Gröbner basis computation in $n + 1$ variables. Another solution is due to Sturmfels' [8] which involves n Gröbner basis computations in n variables.

There are also several efficient algorithms for saturating a pure difference binomial ideal in $k[x_1, \ldots, x_n]$. These were developed in the context of the computation of toric ideals. Computation of a toric ideal has some useful applications including solving integer programs [3,11,10], computing primitive partition identities [8] (chapters 6 and 7), and solving scheduling problems [9].

Urbanke [1] proposed an algorithm which involves of $O(n)$ Gröbner basis computations, all in an n-variable ring, which is similar to Sturmfels' algorithm. Bigatti et.al. [2] proposed an algorithm to compute toric ideal but it is not based on saturation computation.

One important fact about Buchberger's algorithm for computing Gröbner basis is that it is very sensitive to the number of variables of the ring. In all of the algorithms cited so far, all of the Gröbner basis computations have been done in the original ring having n variables. Recently there have been attempts that saturate any pure difference binomial ideal in which computation occurs in rings with fewer indeterminates than in the original ring. Hemmecke and Malkin [6] have proposed a new approach, called *Project and Lift*, which involves the computation of one Gröbner basis in a ring of j variables for $j = 1, 2, \ldots, n$. Their algorithm shows significant improvement over the prevailing best algorithms. Another approach which also attempts at computation in smaller rings is by Kesh and Mehta [7] which also requires the computation of one Gröbner basis in $k[x_1, \cdots, x_j]$ for each j.

1.3 Our Approach

Before proceeding, we will need a few notations. U_i will denote the multiplicatively closed set $\{x_1^{a_1} \cdots x_{i-1}^{a_{i-1}} \ : \ a_j \geq 0, 1 \leq j < i\}$. $\prec_i$ will denote a graded reverse lexicographic term order with x_i being the least. $\varphi_i : k[x_1, \ldots, x_n] \rightarrow k[x_1, \ldots, x_n][U_{i-1}^{-1}]$ is the natural localization map $r \mapsto r/1$.

Algorithm 1 describes the saturation algorithm due to Sturmfels [8]. Algorithm 2 describes the proposed algorithm. The primary motivation for the new approach is that the time complexity of Gröbner basis is a strong function of the number of variables. In the proposed algorithm, a Gröbner basis is computed in the i-th iteration in i variables. The algorithm requires the computation of a Gröbner basis over the ring $k[x_1, \ldots, x_n][U_i^{-1}]$. The Gröbner basis over such a ring is not known in the literature. Thus, we propose a generalization of Gröbner basis, called Pseudo Gröbner basis, and appropriately modify the Buchberger's algorithm to compute it.

```
Data: A homogeneous binomial ideal, $I \subset k[\mathbf{x}]$.
Result: $I : (x_1, \ldots, x_n)^\infty$
1 for $i \leftarrow 1$ to $n$ do
2     $G \leftarrow$ Gröbner basis of $I$ w.r.t. $\prec_i$ ;
3     $I \leftarrow \langle \{ f : x_i^\infty \ : \ f \in G \} \rangle$ ;
4 end
5 return $I$ ;
```

Algorithm 1. Sturmfels' Algorithm

```
Data: A homogeneous binomial ideal, $I \subset k[\mathbf{x}]$.
Result: $I : (x_1, \ldots, x_n)^\infty$
1 for $i \leftarrow 1$ to $n$ do
2     $G \leftarrow$ Pseudo Gröbner basis of $\varphi_i(I)$ w.r.t. $\prec_i$ ;
3     $I \leftarrow \langle \{ \varphi_i^{-1}(f : x_i^\infty) \ : \ f \in G \} \rangle$ ;
4 end
5 return $I$ ;
```

Algorithm 2. Proposed Algorithm

1.4 Refined Problem Statement

Let R be a commutative Noetherian ring with unity, and $U \subset R$ be a multiplicatively closed set with unity but without zero. Let the set U^+ be defined as

$$U^+ = \{ \ u \ : \ u \in U, \text{ or } -u \in U, \text{ or } u = 0 \ \} .$$

Let S denote the localization of R w.r.t U, i.e., $S = R[U^{-1}]$. Define a class of binomials, called U-binomials, in the ring $S[x_1, \ldots, x_n]$ (also denoted by $S[\mathbf{x}]$) as follows

$$\frac{u_1}{u_1'}\mathbf{x}^{\alpha_1} + \frac{u_2}{u_2'}\mathbf{x}^{\alpha_2}$$

where $u_i \in U^+, u_i' \in U$ and $\mathbf{x}^{\alpha_i}$ denotes the monomial $x_1^{a_{i1}} \cdots x_n^{a_{in}}$ for $i = 1, 2$.

We will address the problem of efficiently saturating a homogeneous U-binomial ideal w.r.t. all the variables in the ring, namely $x_1, \ldots, x_n$.

For different choices of R and U, this problem reduces to many standard problems found in the literature. If R is a field, then this problem reduces to saturating a binomial ideal in the standard polynomial ring. Restricting R to a field and U to $\{ +1, -1 \}$ considering only pure difference binomials, it reduces to the problem of saturating a pure difference binomial ideal.

The rest of the paper is arranged as follows. Sections 2 and 3 deals with "chain binomials" and "chain sums" for general binomial ideals. Section 4 deals with reductions of a U-binomial by a set of U-binomials. In section 5, we will present the notion of Pseudo Gröbner Basis for $S[\mathbf{x}]$, and a modified Buchberger's algorithm to compute it. In section 6, we present a result similar to Sturmfels' lemma (Lemma 12.1 [8]). The final saturation algorithm is presented in section 7. Finally, in section 8, we present some preliminary experimental results comparing our algorithm applied to toric ideals, to that of Sturmfels' algorithm and Project and Lift.

2 Chain and Chain-Binomial

In this section we shall describe the terminology we will need to work with general binomials.

Symbols $u, v, w, \ldots$ will denote elements of U^+ and $u', v', w', \ldots$ will denote the elements of U. A *term* in the polynomial ring $S[\mathbf{x}]$ is the product of an S element with a monomial, for example, $(r/u')x_1^{a_1} \ldots x_n^{a_n}$ where $r \in R$ and $u' \in U$. For simplicity in the notations, we may also write it as $(r/u')\mathbf{x}^\alpha$, where α represents the vector $(a_1, \ldots, a_n)$. If $r \in U^+$, then we will call it a *U-term.* A *binomial* is a polynomial with at most two terms, i.e., $b = (r_1/u'_1)\mathbf{x}^\alpha + (r_2/u'_2)\mathbf{x}^\beta$. A binomial of the form $\mathbf{x}^\alpha - \mathbf{x}^\beta$ is called a *pure difference binomial.* If both the terms of a binomial are U-terms, then we will call it a *U-binomial.* A U-binomial of the form $(u_1/u'_1)\mathbf{x}^\alpha + (u_2/u'_2)\mathbf{x}^\alpha$ will be called *balanced.* Since U need not be closed under addition, a balanced U-binomial $((u_1/u'_1) + (u_2/u'_2))\mathbf{x}^\alpha$ need not be a U-term in general. A binomial b is said to be *oriented* if one of its terms is identified as *first* (and the other *second*). If b is oriented, then b^{rev} denotes the same binomial with the opposite orientation.

In the above notations, one of the coefficients of a binomial or U-binomial may be zero. Hence the definition of binomials (rep. U-binomials) includes single terms (resp. U-terms). To be able to handle all binomials in a uniform manner, we shall denote a single term $(r/u')\mathbf{x}^\alpha$ as $(r/u')\mathbf{x}^\alpha + (0/1)\mathbf{x}^\square$, where $\mathbf{x}^\square$ is a symbolic monomial. This will help in avoiding to consider a separate case for single terms in some proofs. We shall refer to such binomials as *mono-binomials.* In a term-ordering, $\mathbf{x}^\square$ will be defined to be the least element. Coefficient of $\mathbf{x}^\square$ in every occurrence will be zero.

Definition 1. *A sequence of oriented binomials* $((r_1/u'_1)\mathbf{x}^{\beta_1}b_1, \ldots, (r_q/u'_q)\mathbf{x}^{\beta_q}b_q)$ *(repetition allowed) will be called a* **chain** *if the second term of* $(r_i/u'_i)\mathbf{x}^{\beta_i}b_i$ *cancels the first term of* $(r_{i+1}/u'_{i+1})\mathbf{x}^{\beta_{i+1}}b_{i+1}$*, for each* $1 \le i < q$*. Let* B *be a set of* U*-binomials. If each* b_i *in the chain belongs to* B*, then we will call it a* B**-chain***. The sum of the binomials of the chain (respectively,* B*-chain)* $\tilde{b} = \sum_{i=1}^{q}(r_i/u'_i)\mathbf{x}^{\beta_i}b_i$*, which is itself a binomial, will be called the corresponding* **chain binomial** *(respectively,* B**-chain binomial***). It is the first term of* $(r_1/u'_1)\mathbf{x}^{\beta_1}b_1$ *plus the second term of* $(r_q/u'_q)\mathbf{x}^{\beta_q}b_q$*, since all the intermediate terms get canceled. We will call any two chains* **equivalent** *if their corresponding chain-binomials are the same.*

In the later sections we will be interested in the "shape" of a chain. Given a term ordering we will call a chain **ascending** if the first monomial is (strictly) less than the second monomial in each binomial of the chain with respect to the given term-order. Similarly **descending** chains are defined. Another shape of significant interest is the one in which there are three sections in the chain: first is descending, second is horizontal (all binomials in it are balanced), and the final section is ascending. Any of these sections can be of length zero. Such chains will be called **bitonic**.

Suppose we have a sequence of oriented U-binomials such that the monomial of the second term of the i-th binomial in the sequence is equal to the monomial of first term of the $(i+1)$-st binomial in the sequence. Then we can multiply suitable coefficients to these U-binomials to turn this sequence into a chain such that its chain-binomial is also a U-binomial. Let $(\mathbf{x}^{\beta_1}b_1, \ldots, \mathbf{x}^{\beta_q}b_q)$ be a sequence of oriented U-binomials such that the first $q-1$ binomials are not mono-binomials. Let $\mathbf{x}^{\beta_i} b_i = \mathbf{x}^{\beta_i}((u_i/u_i')\mathbf{x}^{\alpha_{i,1}} + (v_i/v_i')\mathbf{x}^{\alpha_{i,2}})$ where $(u_i/u_i')\mathbf{x}^{\alpha_{i1}}$ is the first term, for each i. Let $\beta_i + \alpha_{i,2} = \beta_{i+1} + \alpha_{(i+1),1}$ for all $1 \le i < q$. Consider the sequence $(\ldots, (d_i/d_i')\mathbf{x}^{\beta_i} b_i, \ldots), 1 \le i \le q$ where $d_1/d_1' = 1/1$ and

$$d_i/d_i' = (-1)^{i-1}(v_1 u_2' v_2 u_3' v_3 \cdots v_{i-1} u_i' / v_1' u_2 v_2' u_3 v_3' \cdots v_{i-1}' u_i),$$

for $i > 1$. It is easy to see that it is a chain of U-binomials and its chain-binomial is the U-binomial $(u_1/u_1')\mathbf{x}^{\alpha_{1,1}} + (d_q/d_q')(v_q/v_q')\mathbf{x}^{\alpha_{q,2}}$ which will be denoted by $\mathtt{B}(\mathbf{x}^{\beta_1} b_1, \ldots, \mathbf{x}^{\beta_q} b_q)$. Note that if b_q is a mono-binomial, then the second term will be $(0/1)\mathbf{x}^{\square}$.

Observation 1. *Let $(\mathbf{x}^{\beta_1} b_1, \ldots, \mathbf{x}^{\beta_k} b_k)$ be a sequence of oriented U-binomials where $b_i \in B$ and none of which are mono-binomials. Furthermore, the second monomial of $\mathbf{x}^{\beta_i} b_i$ and the first monomial of $\mathbf{x}^{\beta_{i+1}} b_{i+1}$ are same for all $1 \le i < k$. Then $\mathtt{B}(\mathbf{x}^{\beta_1} b_1, \ldots, \mathbf{x}^{\beta_k} b_k, \mathbf{x}^{\beta_k} b_k^{rev}, \ldots, \mathbf{x}^{\beta_1} b_1^{rev}) = 0$.*

3 Decomposition into Chains

If B is a finite set of pure difference binomials, then every binomial in $\langle B \rangle$ is a B-chain binomial. This property is used in the computation of a toric ideal. In case B has general binomials this property does not hold. But in the following theorem we will show that in ideals generated by U-binomials every polynomial can be expressed as the sum of some B-chain binomials. This result is used in the proof of theorems 2 and 3. For any polynomial f, $Mon(f)$ will denote the set of monomials in f.

Theorem 1. *Let B be a finite set of U-binomials in $S[\mathbf{x}]$. For every polynomial f in $I = \langle B \rangle$, there exists a set of B-chain binomials $\tilde{b}_i$ such that $f = \sum_i \tilde{b}_i$ where both monomials of every $\tilde{b}_i$ belongs to $Mon(f) \cup \{\mathbf{x}^{\square}\}$.*

Proof. Let $B = \{b_1, \ldots, b_n\}$. Consider an arbitrary polynomial $f \in \langle B \rangle$. So $f = \sum_i (r_i/w_i')\mathbf{x}^{\beta_i} b_{j_i}$ where $(r_i/w_i')\mathbf{x}^{\beta_i} \in S[\mathbf{x}]$, for all i. Define an edge-weighted graph G (multi-edges and loops allowed) representing this expression in the following manner. The vertex set of this graph is the set of distinct monomials in $(r_i/w_i')\mathbf{x}^{\beta_i} b_i$, for all i. Vertices corresponding to $Mon(f) \cup \{\mathbf{x}^{\square}\}$ will be called **terminal vertices**.

There is one edge for each addend binomial in the sum-expression for f. The i-th edge is incident upon the two monomials associated with $\mathbf{x}^{\beta_i} b_i$, if they are distinct. Otherwise it forms a loop on that monomial. Weights are assigned to two halves of each edge separately. Suppose $b_i = (u_i/u_i')\mathbf{x}^{\alpha_{i,1}} + (v_i/v_i')\mathbf{x}^{\alpha_{i,2}}$. Then we associate weight $(r_i/w_i')(u_i/u_i')$ to the end incident on $\mathbf{x}^{\beta_i}\mathbf{x}^{\alpha_{i,1}}$ and weight $(r_i/w_i')(v_i/v_i')$ to the end incident on $\mathbf{x}^{\beta_i}\mathbf{x}^{\alpha_{i,2}}$.

It should be clear from the construction that the sum of end-weights incident upon a non-terminal vertex must be zero. Hence the degree of non-terminal vertices can never be one. Each end-weight incident on $\mathbf{x}^{\square}$ is zero, so their sum is also zero. See example in figure 1.

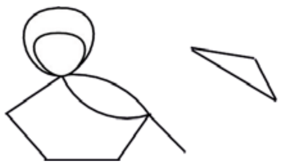


Fig. 1. An example of chain decomposition

Consider any connected component, C, of G. The polynomial corresponding to C is the sum of its monomials, weighted with the sum of end-weights incident on it. This is also the sum of addend binomials corresponding to the edges in C. So the polynomial associated with G is the sum of polynomials of all components of G, which is f.

If a component does not contain any $Mon(f)$ vertex, then the corresponding polynomial will be zero. So we can delete it from the graph without affecting the total polynomial. Similarly any isolated $Mon(f)$ vertex with no loop-edge also contributes zero and can be deleted from the graph. So we can assume that every connected component of G has at least one $Mon(f)$ vertex and degree of all terminal vertices is at least 1 and as observed earlier, the degree of non-terminal vertices is at least 2.

We will establish the claim of the theorem by induction on the number of edges in the graph. If the graph has one edge, then the corresponding expression is a trivial B-chain binomial with both monomials from $Mon(f) \cup \{\mathbf{x}^{\square}\}$. Next we will consider the graphs with more than one edge.

If there is a component with at least two terminal vertices, then select a shortest path w between two different vertices of $Mon(f) \cup \{\mathbf{x}^{\square}\}$ in the component. In case all components have only one $Mon(f)$ node, then from lemma 1 given below, we conclude that a closed walk w exists passing through the terminal vertex and has at least one edge on it which is traversed only once.

In either case, the walk w has at least one edge on it which is not traversed more than once and both its end-vertices (the two end-vertices may be same if w is a closed walk) are terminals. Furthermore, if one of the end-vertices is $\mathbf{x}^{\square}$, then w must be a path, not a closed walk. Hence, all edges on it are traversed only once. In particular, the edge incident on $\mathbf{x}^{\square}$ is traversed only once.

Let $(r_{j_1}/w'_{j_1})\mathbf{x}^{\beta_{j_1}}b_{j_1}, (r_{j_2}/w'_{j_2})\mathbf{x}^{\beta_{j_2}}b_{j_2}, \ldots, (r_{j_k}/w'_{j_k})\mathbf{x}^{\beta_{j_k}}b_{j_k}$ be the sequence of the binomials associated with the successive the edges of the walk. Orient these binomials such that walk proceeds from the first to the second term of each binomial. Then the second monomial of i-th binomial is same as the first monomial of $(i+1)$-st binomial of the walk/sequence.

Suppose the t-th edge in the walk is traversed only once. In case the walk ends in $\mathbf{x}^{\square}$, take t to be the edge incident on $\mathbf{x}^{\square}$. Let $j_t = l$. Consider the chain binomial $\tilde{b} = \mathtt{B}((r_l d'_t/w'_l d_t)\mathbf{x}^{\beta_{j_1}}b_{j_1}, \ldots, (r_l d'_t/w'_l d_t)\mathbf{x}^{\beta_{j_k}}b_{j_k})$, where d_t/d'_t is as defined in the end of section 2. Observe that in the chain expression of $\tilde{b}$ the t-th addend is $(r_l/w'_l)\mathbf{x}^{\beta_l}b_l$ and all the remaining addends correspond to other than l-th edge of the graph. From the definition of binomial $\tilde{b}$, both its monomials are from the set $\{Mon(f) \cup \{\mathbf{x}^{\square}\}$.

Let $f' = f - \tilde{b}$. Express f' as a sum expression by combining the sum expressions of f and $\tilde{b}$. The coefficients of a given binomial in the sum expression of f and of $\tilde{b}$ combine to a single coefficient of the form r/u'. Hence, we get a sum-expression for f' where the binomials are the same as in the expression of f but their coefficients may change. The coefficient of $\mathbf{x}^{\beta_l} b_l$ in f' sum-expression is zero. So the number of addend binomials in f' expression is at least one less that that in f expression. Therefore the graph corresponding to f' will have at least one fewer edge then in the graph of f. This establishes the induction-step and hence the proof is complete. ■

Following is a graph theoretic result which was used in the above theorem.

Lemma 1. *Let H be an undirected connected graph (possibly with loops and multi-edges) with n vertices. Let s be a specified vertex. Also let the degree of every vertex other than s be greater than one and $deg(s) \geq 1$ (so if $n = 1$ then s has a loop). Then, there exists a closed walk passing through s which has at least one edge that occurs only once in it.*

Proof. The number of edges in H is half of the sum of degrees of its vertices, so it is at least $\lceil (1 + 2(n-1))/2 \rceil = n$. A tree on n vertices has $n-1$ edges. So there must exist a cycle in H. Since the graph allows loops and parallel edges, the cycles in the graph include 1-cycles (loop) and 2-cycles (due to parallel edges).

Suppose this cycle is $v_0 \overset{e'_0}{\rightarrow} v_1 \overset{e'_1}{\rightarrow} \ldots v_{m-1} \overset{e'_{m-1}}{\rightarrow} v_0, m \geq 1$. Furthermore, suppose v_i is one of the nearest vertices of the cycle from s and let $e_1, e_2, \ldots, e_t$ be a shortest paths from s to v_i. So this path only touches the cycle at v_i and the sets of the edges of the path and the cycle are disjoint. Then the desired walk is $e_1, e_2, \ldots, e_t, e'_i, e'_{i+1}, \ldots, e'_0, e'_1 \ldots, e'_{i-1}, e_t, e_{t-1}, \ldots, e_1$. ■

4 Reduction of U-Binomials

Let B be a finite set of non-balanced U-binomials (which may include mono-binomials) and a term order $\prec$. In this section we will formally describe the reduction of any U-binomial by B with respect to the given term order. We will assume that each binomial of B is oriented by setting the leading term as the first term. We will denote the leading term of a binomial b by $in_{\prec}(b)$.

Given an arbitrary U-term $(u/u')\mathbf{x}^{\alpha}$, algorithm 1 computes a descending B-chain $(v_1/v'_1)\mathbf{x}^{\beta_1} b_{j_1}, \ldots, (v_p/v'_p)\mathbf{x}^{\beta_p} b_{j_p}$ with corresponding B-chain binomial $\sum_{i=1}^{p} (v_i/v'_i)\mathbf{x}^{\beta_i} b_{j_i} = (u/u')\mathbf{x}^{\alpha} - (w/w')\mathbf{x}^{\gamma}$ where $\mathbf{x}^{\gamma}$ is not divisible by the leading term of any member of B. The term $(w/w')\mathbf{x}^{\gamma}$ will be denoted by $\overline{(u/u')\mathbf{x}^{\alpha}}^{B}$.

Any initial sub-chain $(v_1/v'_1)\mathbf{x}^{\beta_1} b_{j_1}, \ldots, (v_p/v'_p)\mathbf{x}^{\beta_p} b_{j_l}$ is called a **reduction** of $(u/u')\mathbf{x}^{\alpha}$ and if the corresponding chain-binomial is $(u/u')\mathbf{x}^{\alpha} - (w_1/w'_1)\mathbf{x}^{\gamma_1}$, then $(u/u')\mathbf{x}^{\alpha}$ is said to have B-**reduced** to $(w_1/w'_1)\mathbf{x}^{\gamma_1}$. In particular, $\overline{(u/u')\mathbf{x}^{\alpha}}^{B}$ is the irreducible B-reduction of $(u/u')\mathbf{x}^{\alpha}$. If $p = \sum_i (u_i/u'_i)\mathbf{x}^{\alpha_i}$ and $(w_i/w'_i)\mathbf{x}^{\gamma_i}$ be

```
**Data**: A finite set, $B$, of non-balanced $U$-binomials; a $U$-term $(u/u')\mathbf{x}^\alpha$
**Result**: A $U$-term $(w/w')\mathbf{x}^\gamma$ which is irreducible by $B$ and a $B$-chain
        corresponding to binomial $(u/u')\mathbf{x}^\alpha - (w/w')\mathbf{x}^\gamma$.
1  $(w/w')\mathbf{x}^\gamma := (u/u')\mathbf{x}^\alpha$ ;
2  $i := 0$ ;
3  while some leading monomial in $B$ divides $\mathbf{x}^\gamma$ do
4      select $b = (\mu_1/\mu_1')\mathbf{x}^{\delta_1} + (\mu_2/\mu_2')\mathbf{x}^{\delta_2}$ from $B$ s.t. the leading monomial $\mathbf{x}^{\delta_1}$
       divides $\mathbf{x}^\gamma$ ;
5      $i := i + 1$ ;
6      $\mathbf{x}^{\beta_{j_i}} := \mathbf{x}^\gamma/\mathbf{x}^{\delta_1}$ ;
7      $v_i/v_i' := (-w/w')(\mu_1'/\mu_1)$ ;
8      $w/w' := (v_i/v_i')(\mu_2/\mu_2')$;
9      $\mathbf{x}^\gamma := \mathbf{x}^{\beta_{j_i}} \cdot \mathbf{x}^{\delta_2}$;
10 end
11 return $(w/w')\mathbf{x}^\gamma$, $((v_1/v_1')\mathbf{x}^\beta b_{j_1}, \ldots, (v_i/v_i')\mathbf{x}^{\beta_i} b_{j_i})$ ;
```

Algorithm 3. Division algorithm for a U-monomial by a set of non-balanced U-binomials

some B-reduction of $(u_i/u_i')\mathbf{x}^{\alpha_i}$ for each i, then $\sum_i (w_i/w_i')\mathbf{x}^{\gamma_i}$ is a B-reduction of p. The reduction of binomials is of special interest here. Suppose we have a non-balanced U-binomial $b = (u_1/u_1')\mathbf{x}^{\alpha_1} + (u_2/u_2')\mathbf{x}^{\alpha_2}$ and a finite set B of non-balanced U-binomials in which the first term is greater than the second term. Let $(w_1/w_1')\mathbf{x}^{\gamma_1}$ and $(w_2/w_2')\mathbf{x}^{\gamma_2}$ be the reductions of $(u_1/u_1')\mathbf{x}^{\alpha_1}$ and $(u_2/u_2')\mathbf{x}^{\alpha_2}$ respectively. So $b' = (w_1/w_1')\mathbf{x}^{\gamma_1} + (w_2/w_2')\mathbf{x}^{\gamma_2}$ is a reduction of b. Adjoining the reduction chain of $(u_1/u_1')\mathbf{x}^{\alpha_1}$ with b' (if it is non-zero) followed by the reverse of the reduction chain of $(u_2/u_2')\mathbf{x}^{\alpha_2}$ results into a bitonic chain called a **reduction chain** of b with respect to B. Obviously, its chain-binomial is b.

In case b is a balanced U-binomial $(u_1/u_1')\mathbf{x}^\alpha + (u_2/u_2')\mathbf{x}^\alpha$, we only need to reduce $\mathbf{x}^\alpha$. Let a reduction chain and the reduction monomial be C_1 and $(w_1/w_1')\mathbf{x}^\gamma$ respectively. Then $b' = (u_1/u_1')(w_1/w_1')\mathbf{x}^\gamma + (u_2/u_2')(w_1/w_1')\mathbf{x}^\gamma$ is a B-reduction of b and the corresponding reduction chain is $(u_1/u_1')C_1, b', (u_2/u_2')C_1^{rev}$.

For any binomial b, any B-reduction chain which reduces it to b', is a $B \cup \{b'\}$-chain and it is bitonic. In particular, if b' is zero then the reduction chain will be a B-chain.

Lemma 2. *Let C be a B-chain and $b \in B$. Let $B' = B \setminus \{b\}$ and b' be some B'-reduction of b. Then there is a $B' \cup \{b'\}$-chain which is equivalent to C.*

Proof. If b does not occur in C, then C is also a $B' \cup \{b'\}$-chain.

The reduction chain of b by B' is a $B' \cup \{b'\}$-chain. In case b occurs in C, plug this reduction chain in places of b in C. So the resulting chain is equivalent to C and itself a $B' \cup \{b'\}$-chain. ■

5 Pseudo-Gröbner Basis

In the first section we saw that the saturation of an ideal in $k[\mathbf{x}]$ can be computed by first computing a suitable Gröbner basis for it, as described in Sturmfels' lemma (Lemma 12.1 [8]). Unfortunately, Gröbner basis is only defined for ideals in $k[\mathbf{x}]$, where k is a field, not for $S[\mathbf{x}]$ as is the case here. In this section, we will describe a type of basis for U-binomial ideals in $S[\mathbf{x}]$ which closely resembles a Gröbner basis. In section 6, we will also prove a theorem similar to the Sturmfels' lemma which will allow us to compute the saturation of such ideals.

Definition 2. *For every finite U-binomial set G, G_1 and G_2 will denote its partition, where the former will represent the set of non-balanced binomials and the latter will represent the set of balanced binomials of G.*

Definition 3. *Let $b_1 = (u_1/u_1')\mathbf{x}^{\alpha_1} + (v_1/v_1')\mathbf{x}^{\beta_1}$ and $b_2 = (u_2/u_2')\mathbf{x}^{\alpha_2} + (v_2/v_2')\mathbf{x}^{\beta_2}$ be non-balanced U-binomials belonging to $S[\mathbf{x}]$. Let $\prec$ be a term order and $\mathbf{x}^{\beta_i} \prec \mathbf{x}^{\alpha_i}$ for $i = 1, 2$. Further, let $b_3 = (w_1/w_1' + w_2/w_2')\mathbf{x}^{\alpha}$. We define two types of S-**binomials** as follows: First one for a pair of two non-balanced binomials, $s(b_1, b_2)$, is given by $(u_1v_2/u_1'v_2')\mathbf{x}^{\beta_2+\gamma-\alpha_2} - (v_1u_2/v_1'u_2')\mathbf{x}^{\beta_1+\gamma-\alpha_1}$, where $\mathbf{x}^{\gamma}$ is the LCM of $\mathbf{x}^{\alpha_1}$ and $\mathbf{x}^{\alpha_2}$. The second type is for a balanced and non-balanced binomial. In this case $s(b_3, b_1)$ is given by $(w_1/w_1' + w_2/w_2')\mathbf{x}^{\beta_1+\gamma-\alpha_1}$, where $\mathbf{x}^{\gamma}$ is the LCM of $\mathbf{x}^{\alpha}$ and $\mathbf{x}^{\alpha_1}$.*

Assume a fixed term-order. In a chain $(\ldots, (v_i/v_i')\mathbf{x}^{\beta_i} b_i, \ldots)$, two consecutive binomials will be said to form a **peak** if at least one is non-balanced and the monomial at their junction is greater than or equal to the other two monomials. Further suppose $\mathbf{x}^{\beta_{i-1}} b_{i-1}$ and $\mathbf{x}^{\beta_{i+j}} b_{i+j}$ are non-balanced binomials and all the intermediate binomials are balanced, then the binomials $\mathbf{x}^{\beta_k} b_k$, $i \le k \le i+j-1$ are called **plateau** if at least one of $(i-1)$-st and i-th binomials or $(i+j-1)$-th and $(i+j)$-th binomials form a peak. See figure 2.

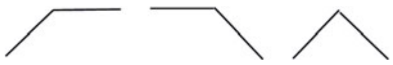


Fig. 2. Types of peaks

Suppose $C = (\ldots, (u_{i-1}/u_{i-1}')\mathbf{x}^{\beta_{i-1}} b_{i-1}, (u_i/u_i')\mathbf{x}^{\beta_i} b_i, \ldots)$ is a chain where $(i-1)$-st and i-th binomials form a peak. In case b_{i-1} and b_i both are non-balanced, then there exists a term $(w/w')\mathbf{x}^{\gamma}$ such that following chain is equivalent to C: $\ldots, (u_{i-2}/u_{i-2}')\mathbf{x}^{\beta_{i-2}} b_{i-2}, (w/w')\mathbf{x}^{\gamma} s(b_{i-1}, b_i), (u_{i+1}/u_{i+1}')\mathbf{x}^{\beta_{i+1}} b_{i+1}$, $\ldots$. In the second case, when b_{i-1} is balanced and b_i is non-balanced, then there exists a constant w_1/w_1' and a term $(w_2/w_2')\mathbf{x}^{\gamma}$ such that the following chain is equivalent to C: $\ldots, (u_{i-2}/u_{i-2}')\mathbf{x}^{\beta_{i-2}} b_{i-2}, (w_1/w_1')\mathbf{x}^{\beta_i} b_i, (w_2/w_2')\mathbf{x}^{\gamma}$ $s(b_{i-1}, b_i), (u_{i+1}/u_{i+1}')\mathbf{x}^{\beta_{i+1}} b_{i+1}, \ldots$. The third case where b_{i-1} is non-balanced and b_i is balanced, need not be separately considered because it is same as the second case with initial chain reversed. Observe that in these cases the original peak is removed, see figure 3.

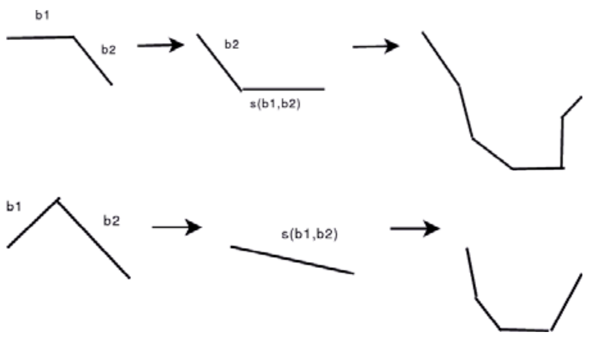


Fig. 3. S-polynomial reductions

Lemma 3. *Let G be a finite set of U-binomials and assume a fixed term-ordering. If for every S-polynomial $s(b_1, b_2)$, $b_1, b_2 \in G$, has a G-chain in which each monomial is less than or equal to at least one monomial of $s(b_1, b_2)$, then every G-chain has an equivalent bitonic G-chain.*

Proof. Consider any arbitrary G-chain. If it has no peak, then it must be bitonic. Otherwise locate one of the highest (in terms of the ordering) peaks. Replace the two binomials forming the peak by the S-polynomial or the combination of the S-polynomial and the non-balanced binomial as described in the previous paragraph. Now replace the S-binomial by the corresponding G-reduction chain. The reduction chain cannot have any monomial higher than both the monomials of the S-binomial so no new peaks can form which is above both the monomials of S-binomial. Substitution again turns the chain into a G-chain and it is equivalent to the original chain. But it has one less peak or plateau at the level of the selected peak. Iterate over this step till there is no peak left. Since term-ordering is well-ordering, these iterations will have to terminate. ■

A functional definition of Gröbner basis for any ideal in the ring $k[\mathbf{x}]$ is that it is a basis of the ideal which reduces every member of the ideal to zero. We will define *pseudo Gröbner basis* in a similar fashion. In the previous section we described the reduction of a U-binomial by a set of non-balanced U-binomials. Hence the reduction of a U-binomial by set G_1 is well defined.

Definition 4. *A U-binomial basis G of the ideal $I = \langle G \rangle$ will be called* pseudo Gröbner basis *with respect to a given term-order if every binomial of I reduces to $0 (mod \langle G_2 \rangle)$.*

Algorithm 4 is modified Buchberger's algorithm which computes a pseudo Gröbner basis for the ideal generated by an initial basis B, containing U binomials. The first loop of the algorithm terminates since the initial ideal of $\langle G_1 \rangle$ strictly increases in each iteration and the underlying ring is Noetherian. In the second part the S-polynomial computed in line 5 and the reduction with respect to G_1 do not change the coefficient of the monomial in the balanced binomial.

Hence all members of H have the same coefficient. In line 5 r' reduces to zero if any monomial in H divides it else it remains unchanged. Therefore each addition to H strictly increases the ideal generated by H. Once again ring being Noetherian, this expansion of H must stop. Hence the algorithm terminates.

```
   **Data**: $B = \{ b_1, \ldots, b_s \} \subseteq S[x_1, \ldots, x_n]$ be a set of $U$-binomials ; a term
             order $\prec$
   **Result**: A pseudo Gröbner basis $(G_1, G_2)$ for $\langle B \rangle$ with respect to $\prec$.
 1 $G_2 \leftarrow$ balanced members of $B$ ;
 2 $G_1 \leftarrow B \setminus G_2$ ;
 3 repeat
 4     $G_{1,old} \leftarrow G_1$ ;
 5     for each pair $b_1, b_2 \in G_{1,old}$ s.t. $b_1 \neq b_2$ do
 6         $r \leftarrow \overline{s(b_1, b_2)}^{G_1}$ ;
 7         if $r$ is non-balanced then
 8             $G_1 \leftarrow G_1 \cup \{r\}$ ;
 9         else
10             if $r \neq 0$ then
11                 $G_2 \leftarrow G_2 \cup \{r\}$
12             end
13         end
14     end
15 until $G_{1,old} = G_1$;
16 $H_2 \leftarrow \emptyset$;
17 for each $b$ in $G_2$ do
18     $H \leftarrow \{b\}$;
19     repeat
20         $H_{old} \leftarrow H$;
21         for each $b \in H_{old}$ and $b_1 \in G_1$ do
22             $r' \leftarrow \overline{s(b, b_1)}^{G_1}$;
23             $r \leftarrow \overline{r'}^{H}$;
24             if $r \neq 0$ then
25                 $H \leftarrow H \cup \{r\}$;
26             end
27         end
28     until $H_{old} = H$;
29     $H_2 \leftarrow H_2 \cup H$;
30 end
31 $G_2 \leftarrow H_2$;
   /* For reduced pGB, reduce $G_1$ elements by other $G_1$ elements
      and $G_2$ elements by $G_1$.                                   */
32 return $(G_1, G_2)$;
```

Algorithm 4. $\mathcal{A}_1$: Modified Buchberger's algorithm

Theorem 2. *Algorithm 4 computes a pseudo Gröbner basis of $\langle B\rangle$ with respect to the given term ordering.*

Proof. Let (G_1, G_2) be the output of algorithm 4. Let $G = G_1 \cup G_2$. The S-polynomials of a pair of binomials in the ideal also belong to the ideal. Similarly the G_1 reduction of a binomial of the ideal also belongs to the ideal. Hence the ideal remains fixed during the computation, i.e., $\langle B\rangle = \langle G\rangle$.

In order to show that (G_1, G_2) is a pseudo-Gröbner basis of $\langle G\rangle$ we need to show that G_1 reduces every polynomial of $\langle G\rangle$ to polynomial in $\langle G_2\rangle$. Due to theorem 1 it is sufficient to show that G_1 reduces every G-chain binomial to a polynomial in $\langle G_2\rangle$.

Let $s(b_1, b_2)$ be the S-polynomial of some $b_1, b_2 \in G$. Then it is itself a $G \cup \{s(b_1, b_2)\}$-chain (i.e., a chain of only one binomial). The reduction chain of $s(b_1, b_2)$ is a G-chain since $\overline{s(b_1, b_2)}^{G_1}$ belongs to G. From Lemma 3 every G-chain has an equivalent bitonic G-chain.

Consider an arbitrary G-chain binomial $b = (u_1/u_1')\mathbf{x}^{\alpha_1} + (u_2/u_2')\mathbf{x}^{\alpha_2}$. From the previous paragraph we know that there is a bitonic G-chain with b as its chain binomial. Let C_1, C_2 and C_3 be its descending, horizontal, and ascending sections. So the C_1 and C_3^r (reverse of C_3) are reduction chains of $(u_1/u_1')\mathbf{x}^{\alpha_1}$ and $(u_2/u_2')\mathbf{x}^{\alpha_2}$ respectively. Let their reduced terms be $(v_1/v_1')\mathbf{x}^{\beta_1}$ and $(v_2/v_2')\mathbf{x}^{\beta_2}$ respectively. Then the chain-binomial of C_2 is $b' = (-v_1/v_1')\mathbf{x}^{\beta_1} + (-v_2/v_2')\mathbf{x}^{\beta_2}$. Since all balanced binomials of G belong to G_2, C_2 is a G_2-chain and $b' \in \langle G_2\rangle$. ∎

6 Saturation with Respect to x_i

In this section we we will prove a result similar to lemma 12.1 of [8] which will result into an algorithm to compute $\langle B\rangle : x_i^\infty$ efficiently.

Theorem 3. *Let (G_1, G_2) be the pseudo Gröbner basis of a homogeneous U-binomial ideal I in $S[\mathbf{x}]$ with respect to graded reverse lexicographic term order with x_i least. Then $(G_1' = G_1 \div x_i^\infty, G_2' = G_2 \div x_i^\infty)$ is a pseudo Gröbner basis of $I : x_i^\infty$.*

Proof. From theorem 1 we know that every polynomial f in I can be expressed as a sum of G-chain binomials and their monomials are monomials of f. So it is sufficient to show that for each G-chain binomial b, $b' = b \div x_i^\infty$ is a G'-chain binomial.

Let $b = (u_1/u_1')\mathbf{x}^{\alpha_1} + (u_2/u_2')\mathbf{x}^{\alpha_2}$ be a G-chain binomial. From lemma 3 there is a bitonic G-chain for b, say, $(v_1/v_1')\mathbf{x}^{\beta_1}b_1, \ldots, (v_k/v_k')\mathbf{x}^{\beta_k}b_k$. Hence every monomial in the chain is less than either $\mathbf{x}^{\alpha_1}$ or $\mathbf{x}^{\alpha_2}$. Let a be the largest integer such that x_i^a divides b, i.e., x_i^a divides $\mathbf{x}^{\alpha_1}$ and $\mathbf{x}^{\alpha_2}$. Since the term ordering is graded reverse lexicographic with x_i least, x_i^a must divide every monomial of the chain. Hence there exists β_j' such that $(\mathbf{x}^{\beta_j}b_j) \div x_i^a = \mathbf{x}^{\beta_j'}(b_j \div x_i^\infty)$. So $b \div x_i^\infty = b \div x_i^a = \sum_j (v_j/v_j')\mathbf{x}^{\beta_j'}(b_j \div x_i^\infty)$ and $(v_1/v_1')\mathbf{x}^{\beta_1'}(b_1 \div x_i^\infty), \ldots, (v_k/v_k')\mathbf{x}^{\beta_k'}(b_k \div x_i^\infty)$ is a chain with chain-binomial equal to $b \div x_i^\infty$. Thus $b \div x_i^\infty$ is a G'-chain binomial. ∎

7 Final Algorithm

Let R_0 be a commutative Noetherian ring with unity, and $U_0 \subset R_0$ be a multiplicatively closed set with unity but without zero. Let the set U_0^+ be defined as

$$U_0^+ = \{ \ u \ : \ u \in U_0, \text{ or } -u \in U_0, \text{ or } u = 0 \ \}.$$

Let S_0 denote the localization of R_0 w.r.t U_0, i.e., $S_0 = R_0[U_0^{-1}]$. Here we define a few notations to simplify the description of the algorithm. Let U_i be the set of all monomials in $x_1, \ldots, x_i$ and $S_i = S_0[x_1, \ldots, x_i][U_i^{-1}]$.

Let $f(\mathbf{x})$ be a polynomial in $S_i[x_{i+1}, \ldots, x_n]$. Let k be the largest integer such that x_i^k occurs in the denominators of one or more terms of f. Then $x_i^\infty * f(\mathbf{x})$ denotes $x_i^k * f(\mathbf{x})$. If B is a set of polynomials of $S_i[x_{i+1}, \ldots, x_n]$, then $x_i^\infty * B$ denotes $\{ \ x_i^\infty * f(\mathbf{x}) \ : \ f(\mathbf{x}) \in B \ \}$.

We will be dealing with several polynomial rings simultaneously. In case of ambiguity about the underlying ring we will denote the ideal generated by a set of polynomials B in a ring $S[\mathbf{x}]$ by $\langle B \rangle_{S[\mathbf{x}]}$.

Our algorithm is based on the following identities where B is a finite set of polynomials in $S_0[x_1, \ldots, x_n]$ and for each i, B_i denotes a basis of $\langle B \rangle_{S_n} \cap S_i[x_{i+1}, \ldots, x_n]$.

Lemma 4. *(i)* $\langle B \rangle_{S_0[x_1,\ldots,x_n]} : (x_1 \ldots x_n)^\infty = \langle B \rangle_{S_n} \cap S_0[x_1, \ldots, x_n]$
(ii) $\langle B \rangle_{S_n} \cap S_{i-1}[x_i, \ldots, x_n] = \langle x_i^\infty * B_i \rangle_{S_{i-1}[x_i,\ldots,x_n]} : (x_i)^\infty$

Proof. (i) Let $f \in \langle B \rangle_{S_n} \cap S_0[x_1, \ldots, x_n]$ so $f = \sum_j (r_j/u_j')(x^{\alpha_j}/\mathbf{x}^{\beta_j})b_j$ where $b_j \in B$. The terms in the denominator in expression get canceled since f has no terms in the denominator. So

$$\mathbf{x}^{\beta_1+\beta_2+\cdots}.f = \sum_j \mathbf{x}^{\alpha_j+\beta_1\cdots+\beta_{j-1}+\beta_{j+1}+\cdots}.b_j \in \langle B \rangle_{S_0[x_1,\ldots,x_n]}.$$

Therefore $f \in \langle B \rangle_{S_0[x_1,\ldots,x_n]} : (x_1 \ldots x_n)^\infty$.

Conversely, Let $f \in \langle B \rangle_{S_0[x_1,\ldots,x_n]} : (x_1 \ldots x_n)^\infty$. So for some $\mathbf{x}^\beta$, $\mathbf{x}^\beta f = \sum_i (r_i/u_i')\mathbf{x}^{\alpha_i} b_i$ where $b_i \in B$. So $f = \sum_i (\mathbf{x}^{\alpha_i}/\mathbf{x}^\beta) b_i \in \langle B \rangle_{S_n}$. Since f has no terms in the denominators of its terms, $f \in \langle B \rangle_{S_n} \cap S_0[x_1, \ldots, x_n]$.

(ii) Let $f \in \langle B \rangle_{S_n} \cap S_{i-1}[x_i, \ldots, x_n]$. So $f \in \langle B_i \rangle$. Let $f = \sum (\mathbf{x}^{\alpha_j}/\mathbf{x}^{\beta_j}) b_j$ where $b_j \in B_i$ and $\mathbf{x}^{\beta_j}$ are monomials on $x_i, x_{i+1}, \ldots$. Let m be the largest exponent of x_i in the denominators in the sum-expression. So there are integers t_i such that $x_i^m f = \sum (x_i^{t_i} \mathbf{x}^{\alpha_j}/\mathbf{x}^{\beta_j})(x_i^\infty * b_j)$. This sum belongs to $\langle x_i^\infty * B_i \rangle_{S_{i-1}[x_i,\ldots,x_n]}$. So $f \in \langle x_i^\infty * B_i \rangle_{S_{i-1}[x_i,\ldots,x_n]} : (x_i)^\infty$.

Now the converse. $x_i^\infty * B_i \subset \langle B \rangle_{S_n} \cap S_{i-1}[x_i, \ldots, x_n]$. So $\langle x_i^\infty * B_i \rangle_{S_{i-1}[x_i,\ldots,x_n]} \subset \langle B \rangle_{S_n} \cap S_{i-1}[x_i, \ldots, x_n]$. Now we will show that the ideal on the right hand side is saturated with respect to x_i. Let $x_i^k f \in \langle B \rangle_{S_n} \cap S_{i-1}[x_i, \ldots, x_n]$ where $x_i, \ldots, x_n$ are not in the denominators in f. So $(1/x_i^k)(x_i^k f) \in \langle B \rangle_{S_n}$ or $f \in \langle B \rangle_{S_n}$. Since f does not have $x_i, \ldots, x_n$ in the denominators, $f \in \langle B \rangle_{S_n} \cap S_{i-1}[x_i, \ldots, x_n]$. ■

Using Theorem 3 we compute the saturation $\langle x_i^\infty * B_i \rangle_{S_{i-1}[x_i,\ldots,x_n]} : (x_i)^\infty$. Hence the final algorithm is as follows.

```
Data: Finite set $B$ of homogeneous $U_0$-binomials in $S_0[x_1,\dots,x_n]$.
Result: A pseudo-Gröbner basis of $\langle B\rangle_{S_0[x_1,\dots,x_n]} : (x_1\cdots x_n)^\infty$
1  $G_1 := \{b \in B | b$ is non-balanced $\}$;
2  $G_2 := \{b \in B | b$ is balanced $\}$;
3  for $i \leftarrow n$ to $1$ do
4      if $i > 1$ then
5          Homogenize $G_1$ using a new variable $z$;
6      end
7      $(G_1', G_2') := (x_i^\infty * G_1, x_i^\infty * G_2)$;
8      $(G_1, G_2) := \mathcal{A}_2(G_1, G_2,$ rev. lex order with $i$ least $)$;
9      $(G_1, G_2) := (G_1 \div x_i^\infty, G_2) \div x_i^\infty)$;
10     $(G_1, G_2) := (G_1|_{z=1}, G_2|_{z=1})$;
11 end
12 return $(G_1, G_2)$.
```

Algorithm 5. $\mathcal{A}_3$:Computation of $\langle B\rangle_{S_0[x_1,\dots,x_n]} : (x_1\cdots x_n)^\infty$

The graded reverse lexicographic term order requires a homogeneous ideal, hence we require homogenization for $n \geq i > 1$ cases. In case of $i = 1$, the ideal is given to be homogeneous.

Theorems 2, 3 and lemma 4 establish the correctness of this algorithm.

Theorem 4. *Let R_0 be Noetherian commutative ring with unity. Let $U_0 \subset R_0$ be a multiplicatively closed set. Let B be a finite set of homogeneous U_0-binomials in $S_0[x_1,\dots,x_n]$. Then algorithm $\mathcal{A}_3$ computes a pseudo-Gröbner basis of $\langle B\rangle : (x_1\cdots x_n)^\infty$.*

8 Preliminary Experimental Results

In the table given below, we present some preliminary experimental results of the application of the proposed algorithm in computing toric ideals. To apply our general algorithm to this specific case, we choose R_0 to be a field k, and U_0 to be $\{1\}$. Thus, $S_0 = k$ and the polynomial ring $S_0[\mathbf{x}]$ is simply $k[\mathbf{x}]$.

We compare our algorithm with Sturmfels' [8] and the Project and Lift algorithm [6], the best algorithm known to date to compute toric ideals. As expected, the table shows that our algorithm performs much better than the Sturmfels' original algorithm, as our algorithm is specifically designed for binomial ideals.

To compare with Project and Lift algorithm, we implemented it as reported on page 19 of [6], without optimizations reported subsequently. 4ti2[6] is the optimal implementation of their algorithm. Similar optimizations are applicable in our algorithm and it too is implemented without the same. The typical results are presented in the table given below.

Our intuition as to why our algorithm is doing better compared to Project and Lift is that, though Project and Lift does a large part of its calculations in rings of variables less than n, it still uses Sturmfels' saturation algorithm

Table 1. Preliminary experimental results comparing Project-and-Lift and our proposed algorithm

Number of	Size of basis		Time taken (in sec.)		
variables	Initial	Final	Sturmfels'	Project and Lift	Proposed
8	4	186	.30	0.12	0.10
	6	597	2.61	.6	0.64
10	6	729	3.2	1.1	0.50
	8	357	2.4	.40	0.29
12	6	423	1.7	.90	0.27
	8	2695	305	60	27.2
14	10	1035	10.5	4.2	2.5

as a subroutine, though the extent it uses the algorithm depends on the input ideal. On the other hand, our algorithm computes all saturations by the same approach.

References

1. Biase, F.D., Urbanke, R.: An algorithm to calculate the kernel of certain polynomial ring homomorphisms. Experimental Mathematics 4, 227–234 (1995)
2. Bigatti, A.M., Scala, R., Robbiano, L.: Computing toric ideals. J. Symb. Comput. 27(4), 351–365 (1999)
3. Conti, P., Traverso, C.: Büchberger algorithm and integer programming. In: AAECC, pp. 130–139 (1991)
4. Cox, D.A., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics). Springer-Verlag New York, Inc., Secaucus (2007)
5. Eisenbud, D., Sturmfels, B.: Binomial ideals. Duke Mathematical Journal 84(1), 1–45 (1996)
6. Hemmecke, R., Malkin, P.N.: Computing generating sets of lattice ideals and markov bases of lattices. Journal of Symbolic Computation 44(10), 1463–1476 (2009)
7. Kesh, D., Mehta, S.K.: Generalized reduction to compute toric ideals. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 483–492. Springer, Heidelberg (2009)
8. Sturmfels, B.: Gröbner Bases and Convex Polytopes. University Lecture Series, vol. 8. American Mathematical Society, Providence (December 1995)
9. Tayur, S.R., Thomas, R.R., Natraj, N.R.: An algebraic geometry algorithm for scheduling in the presence of setups and correlated demands. Mathematical Programming 69(3), 369–401 (1995)
10. Thomas, R.R., Weismantel, R.: Truncated Gröbner bases for integer programming. In: Engineering, Communication and Computing, pp. 241–257. Kluwer Academic Publishers, Dordrecht (1995)
11. Urbaniak, R., Weismantel, R., Ziegler, G.M.: A variant of the Büchberger algorithm for integer programming. SIAM J. Discret. Math. 10(1), 96–108 (1997)

Improved Algorithms for Farthest Colored Voronoi Diagram of Segments[⋆]

Yongding Zhu and Jinhui Xu

Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, NY 14260, USA
{yzhu3,jinhui}@buffalo.edu

Abstract. Given n line segments in the plane with each colored by one of k colors, the Farthest Colored Voronoi Diagram (FCVD) is a subdivision of the plane such that the region of a c-colored site (segment or subsegment) s contains all points of the plane for which c is the farthest color and s is the nearest c-colored site. FCVD is a generalization of the Farthest Voronoi Diagram (i.e., $k = n$) and the regular Voronoi Diagram (i.e., $k = 1$). In this paper, we first present a simple algorithm to solve the general FCVD problem in an output-sensitive fashion in $O((kn + I)\alpha(H) \log n)$ time, where I is the number of intersections of the input and H is the complexity of the FCVD. We then focus on a special case, called Farthest-polygon Voronoi Diagram (FPVD), in which all colored segments form k disjoint polygonal structures (i.e., simple polygonal curves or polygons) with each consisting of segments with the same color. For FPVD, we present an improved algorithm with a running time of $O(n \log^2 n)$. Our algorithm has better performance and is simpler than the best previously known $O(n \log^3 n)$-time algorithm.

1 Introduction

In this paper, we consider the following *Farthest Colored Voronoi Diagram* (FCVD) problem: Given k sets of line segments $S_1, S_2, \ldots, S_k$ in the plane with each S_i associated with a different color c_i, compute a subdivision of the plane such that the region (called *Voronoi region*) of a site $s_{ij} \in S_i$ contains all points of the plane for which S_i is the farthest set and s_{ij} is the nearest site in S_i, where $S_i = \{s_{i1}, s_{i2}, \ldots, s_{in_i}\}$ for $i = 1, 2, \ldots, k$ and $\sum_{i=1}^{k} n_i = n$. For simplicity, we assume that all segments in each set do not intersect with each other in their interiors (i.e., they could share endpoints but not interior points), and thus can be viewed as a collection of open segments and endpoints. This assumption is valid since we can otherwise partition each segment into subsegments at its intersection points with others in the same set. (Note that segments in different sets could still intersect in their interiors.) When $k = 1$, FCVD is known as the

[⋆] This research was partially supported by NSF through CAREER Award CCF-0546509 and grant IIS-0713489.

(Closest) Voronoi Diagram of Segments, and can be computed in $O(n \log n)$ time [4]. When $k = n$, it is the *Farthest Line Segment Voronoi Diagram* and can be computed in $O(n \log n)$ time as well [2].

For arbitrary k, Huttenlocher *et al.* have extensively studied the FCVD problem from a different perspective [6]. Their approach considers the upper envelope of the Voronoi surfaces in 3-space under L_p metrics. For each set S_i, consider the surface represented by the graph of the function $d_i(x) = \min_{s \in S_i} \rho(x, s)$, where ρ is the Euclidean distance and s is an open segment or an endpoint. The surface is called the *Voronoi Surface* of S_i with d_i as the third dimension (see Figure 1). It is well-known that the orthogonal projection of the Voronoi surface renders the Voronoi diagram of S_i. The upper envelope of these surfaces is the graph of the function $f(x) = \max_{1 \le i \le k} d_i(x)$, i.e., $f(x)$ is the largest distance from x to its k nearest sites (open segments or endpoints), one for each set. Then the FCVD can be obtained by projecting the upper envelope to the plane such that each Voronoi region corresponds to a facet of the upper envelope surface. Each such Voronoi region is called a (Voronoi) *cell* and there is a $1-1$ correspondence between Voronoi cells and facets of the Voronoi surface. The upper envelope of k Voronoi surfaces of $S_1, S_2, \ldots, S_k$ under Euclidean distance has a complexity of $O(n^2 2^{\alpha(n)})$ in the worst case and can be computed in $O(n^2 \alpha(n) \log n)$ time, where $\alpha(n)$ is the reverse Ackermann function [6]. It is straightforward to see that our problem can be solved in $O(n^2 \alpha(n) \log n)$ time. In this paper, we present a simple but more efficient algorithm to solve the FCVD problem in $O((kn + I)\alpha(H) \log n)$ time, where I is the number of intersection points of the input and H is the size of the resulting Voronoi diagram.

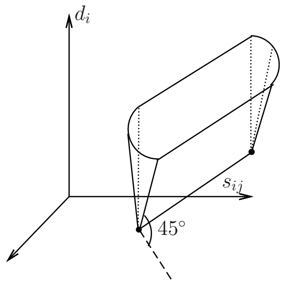


Fig. 1. The Voronoi surface of a (closed) segment consists of a wedge and two half cones; $d_i(x)$ is the lower envelope of such shapes for every site s_{ij} in S_i, where $j = 1, 2, \ldots, n_i$

The main focus of this paper is on a special case of the FCVD problem, called the Farthest-Polygon Voronoi Diagram (FPVD) in which each S_i forms a simple polygon disjoint from all other $k-1$ polygons. For this special case, one way to solve it is to apply Huttenlocher's idea [6]. However, the running time of such an approach would be rather costly for large k (i.e., $O(kn\alpha(n) \log n)$). Recently, Cheong *et al.* designed a more efficient algorithm to solve the problem in $O(n \log^3 n)$ time using a divide-and-conquer paradigm and parametric searching technique. Although parametric searching is a rather powerful technique, it also has some undesirable weakness [1]. For example, parametric searching requires to design an efficient parallel algorithm for the generic version of the decision procedure and exact computation of the polynomials which could be quite complicated and time-consuming. In this paper, we present an alternative approach to solve the problem. We use divide-and-conquer and arc-tracing techniques to generate the farthest-polygon Voronoi diagram in $O(n \log^2 n)$ time. Our algorithm is faster and simpler.

The rest of the paper is organized as follows. In Section 2, we present our algorithm for generating the FCVD for segments. In Section 3, we first modify the algorithm in Section 2 to solve the farthest-polygon Voronoi diagram problem in $O(kn \log n)$ time for small k and then give a new $O(n \log^2 n)$-time algorithm for any k with its running time independent of k. In Section 4, we draw conclusions.

2 Farthest Colored Voronoi Diagram of Segments

Let $S_1, S_2, \ldots, S_k$ be the k sets of segments in the FCVD problem. We consider the regular Voronoi diagram (denoted as $VD(S_i)$) for each set S_i, $i = 1, 2, \ldots, k$. $VD(S_i)$ is constructed using Yap's model [8], where S_i consists of open segments and endpoints. Thus, a site can be either open segment or point in this paper. Two neighboring Voronoi regions, say $VR(s)$ and $VR(s')$, can be separated by a parabolic curve if one of the two sites s and s' is an open segment, the other is a point, and the point site is not an endpoint of the segment site. Without loss of generality, we assume that all sites are in general positions, i.e., there exist at most three sites co-circular.

Definition 1. *A region R in the plane has* star-shape *property if there exists a point $p \in R$ such that for any point $q \in R$ the segment pq lies entirely inside R; R has* weak star-shape *property if there exists a segment s in R such that for any point q of R, let p be the closest point of s to q, then pq lies entirely within R.*

The Voronoi regions have either star-shape or weak star-shape property depending on whether the site is a point or an open segment.

Lemma 1. *Let $VD(S_i)$ be the Voronoi diagram of S_i and $s \in S_i$ be a site. Then the Voronoi region $VR(s)$ has star-shape property if s is a point and weak star-shape property if s is an open segment.*

Proof. As shown in Figure 2, pq lies entirely within the Voronoi region if q is in the region, because for any point q' on the segment pq the circle centered at q' and touches p is contained in the circle centered at q. $\square$

To generate the FCVD, we need to compute the farthest colored Voronoi region $FCVR(s)$ of each site $s \in S_i (i = 1, 2, \ldots, k)$, which can be computed by using the following lemma.

Lemma 2. *Let $VR_i(s)$ be the Voronoi region of $s \in S_i$ in $VD(S_i)$ and $VR_{ij}(s)$ be the Voronoi region of s in $VD(S_i \cup S_j)$ for some $j \neq i$. Then $FCVR(s) = VR_i(s) \setminus \cup_{j \neq i} VR_{ij}(s)$.*

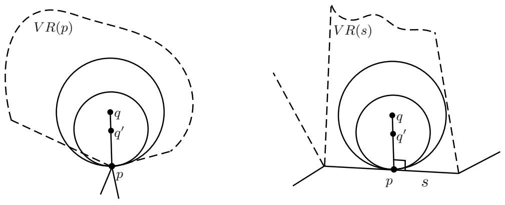


Fig. 2. Star-shaped and weakly star-shaped Voronoi regions; the dashed arcs are Voronoi arcs(i.e., the arcs of the Voronoi diagram separating neighboring Voronoi regions); the left figure demonstrates the case where the site is a point and the Voronoi region is star-shaped; the right figure demonstrates the case where the site is a segment and the Voronoi region is weakly star-shaped

Proof. First we observe that $FCVR(s) \subset VR_i(s)$ and $VR_{ij}(s) \subset VR_i(s)$, where $i, j = 1, 2, \ldots, k$ and $j \neq i$. For any point $q \in VR_{ij}(s)$, q is not in $FCVR(s)$ because q is farther away from S_j than S_i (see Figure 3). □

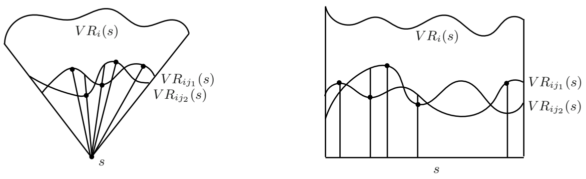


Fig. 3. FCVR of a site s where s can be either point(left) or open segment(right); the boundary of Voronoi regions consists of line segments and parabolic curves; the black dots are vertices

Since by Lemma 1 all Voronoi regions have (weak) star-shape property, we can compute the union of Voronoi regions using the upper envelope algorithms on the boundaries [6]. In particular, for Voronoi regions of a point site, we need to consider the polar coordinate system where the point site is the origin. The upper envelope of line segments and parabolic curves can be computed in $O(n\alpha(h)\log h)$ time in an output-sensitive fashion by the same technique in [7], where h is the complexity of the upper envelope. We are now ready to present the algorithm for computing FCVD.

FCVD Algorithm:
Input: k sets of line segments $S_1, S_2, \ldots, S_k$, each set is painted in unique color;
Output: The Farthest Color Voronoi Diagram of $S_1, S_2, \ldots, S_k$;

- Step 1: Construct Voronoi diagram for each set $VD(S_i)$ where $i = 1, 2, \ldots, k$;
- Step 2: Construct Voronoi diagrams $VD(S_i \cup S_j)$ where $i, j = 1, 2, \ldots, k$ and $j \neq i$;

– Step 3: For each endpoint or subsegment $s \in S_i$, compute $\cup_{j \neq i} VR_{ij}(s)$;
– Step 4: Compute $VR_i(s) \setminus \cup_{j \neq i} VR_{ij}(s)$;

Theorem 1. *The FCVD algorithm correctly generates the FCVD of $S_1, S_2, \ldots, S_k$ in $O((kn + I)\alpha(H)\log n)$ time, where n is the total complexity of the input, I is the number of intersection points, and H is the complexity of the FCVD.*

Proof. The correctness follows from Lemma 2. To analyze the running time, let n_i be the complexity of S_i. Then, we have $n = \sum_{i=1}^{k} n_i$. It takes $O(n_i \log n_i)$ time to construct $VD(S_i)$ [4]. Thus, step 1 takes $O(\sum_{i=1}^{k} n_i \log n_i) = O(n \log n)$ time. Similarly, we can compute $VD(S_i \cup S_j)$ in $O((n_i + n_j + I_{ij}) \log (n_i + n_j + I_{ij}))$ time, where I_{ij} is the total number of intersection points between S_i and S_j. Hence, the total running time for step 2 is at most $\sum_{i=1}^{k}(\sum_{j=1}^{k}(n_i + n_j + I_{ij}) \log n) = \sum_{i=1}^{k}(kn_i + n + \sum_{j=1}^{k} I_{ij}) \log n = (2kn + I) \log n$. Thus the second step takes $O((kn + I) \log n)$ time. Similarly, step 3 takes $O((n + I)\alpha(H) \log H)$ in total, where H denotes the output complexity [7]. The last step can be done in $O(H)$ time by traversing all arcs of $VR_i(s)$ and $\cup_{j \neq i} VR_{ij}(s)$ because both of them have (weak) star-shape property. Thus the total running time of the algorithm is $O((kn + I)\alpha(H) \log n)$. □

3 Farthest-Polygon Voronoi Diagram

Given k sets of line segments $S_1, S_2, \ldots, S_k$ in the plane, the FCVD of the k sets is called a Farthest-polygon Voronoi Diagram (FPVD), if each input set forms a simple polygonal structure (i.e., a polygonal curve or polygon). In this paper, we consider the case that the k polygonal structures are pairwise disjoint. Clearly, by applying the FCVD algorithm, we can construct the FPVD in $O(kn\alpha(H) \log n)$ time as $I = 0$ in this case. If k is a constant, the algorithm is asymptotically near optimal (i.e., differ from the optimal by a factor of $O(\alpha(H))$). However, for large k (e.g., $k = \Omega(n)$), this simple algorithm turns out to be very inefficient. To obtain more efficient solution, we improve the divide-and-conquer approach in [3] and present an algorithm with a running time of $O(n \log^2 n)$. We first introduce some basic concepts and lemmas.

For simplicity, we first assume that there is a large enough bounding box containing all vertices of FPVD in its interior. Thus any unbounded arc of $FPVD$ intersects the bounding box. We call such an intersection point a *bounding-box vertex.*

For a given polygonal structure P, let $CH(P)$ be the region bounded by the convex hull of P (also called the convex hull of P). A *pocket* of P is a connected component of $CH(P) \setminus P$. A pocket can be bounded or unbounded. Let $VD(P)$ be the Voronoi diagram of P. Then it is easy to see that the intersection of $VD(P)$ and a pocket of P is a tree. $VD(P)$ is also called the *medial axis* of P. For unbounded pocket, the tree is rooted at a bounding-box vertex.

A vertex p of FPVD is a point in the plane such that there exists a minimal disk $D(p, r)$ (centered at p and with radius r) that touches at most 3 polygons and intersects all others. By "touch", we mean that the interior of the disk and the polygon are disjoint but there exists at least one point of the polygon lies on the boundary of the disk. Since all sites are in general position, this disk can only touch at most 3 sites. If all the 3 sites belong to the same polygon, the vertex is called a *medial axis vertex.* If each of the 3 sites belongs to a different polygon, the vertex is called a *pure vertex.* Otherwise, the vertex is called a *mixed vertex* [3]. The disk $D(p, r)$ is called the *spanning disk* at p. Minimal spanning disk is unique at any point in the plane.

The arcs of FPVD can also be classified into 3 types. Given two neighboring Voronoi regions of FPVD $VR(s)$ and $VR(s')$ and the arc separating them, the 3 types of arcs are defined as follows:

1. The arc is a *medial axis arc*, if s and s' are in the same set and one is not the endpoint of the other;
2. It is a *pure arc*, if s and s' are in different sets;
3. It is a *spoke*, if s is the endpoint of s' or vice versa.

Lemma 3. *Let $\mathcal{T}$ be a tree of $VD(P)$ and $VR(P)$ be the Voronoi region of P. Then $\mathcal{T} \cap VR(P)$ is a connected subtree.*

Proof. Suppose $\mathcal{T} \cap VR(P)$ is two connected subtrees, say, $\mathcal{T}_1$ and $\mathcal{T}_2$ as shown in Figure 4. Let p and q be two arbitrary points of $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. Then there exists a unique path pq connecting p and q. Since $\mathcal{T}_1$ and $\mathcal{T}_2$ are separated, pq must intersect the boundary of $VR(P)$ at two mixed vertices, denoted by p' and q' respectively. Consider the spanning disks $D_{p'}$ and $D_{q'}$ at p' and q' respectively, then portions of $D_{p'}$, $D_{q'}$, and P enclose a closed region R. In particular, as shown in Figure 4, a and c are the two touching points of P and $D_{p'}$; b and d are the two touching points of P and $D_{q'}$. Then there must exist two input polygons Q and Q' (note that it is possible $Q = Q'$) such that Q touches R at arc ac and Q' touches R at arc bd. Therefore ab and cd must be disconnected because otherwise Q or Q' intersects P. This contradicts the fact that any pair of input polygons are disjoint. Since P is an input polygon and therefore its boundary is connected, we conclude that $\mathcal{T} \cap VR(P)$ must be a connected subtree of $\mathcal{T}$. □

Note that the above lemma was proved in [3]. We give a much simpler proof here for the design of our algorithm.

Given a set $\mathcal{A}$ of k disjoint polygons $S_1, S_2, \ldots, S_k$ with a total complexity of n, it was shown in [3] that one can divide $\mathcal{A}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that the sizes of $\mathcal{A}_1$ and $\mathcal{A}_2$, denoted by n_1 and n_2 respectively, have the following relation: $\frac{n}{4} \leq n_1, n_2 \leq \frac{3n}{4}$ or there exists a polygon S_i whose complexity is larger than or equal to $\frac{n}{2}$. In the latter case, we let $\mathcal{A}_1 = \{S_i\}$ and $\mathcal{A}_2 = \mathcal{A} \backslash \{S_i\}$. We call such a division of $\mathcal{A}$ a *proper division.* Intuitively, if an input set is properly divided, we can apply the divide-and-conquer paradigm in [3] to solve our problem efficiently because the depth of the recursion tree is $O(\log n)$. We can use the following recursive procedure to solve our problem.

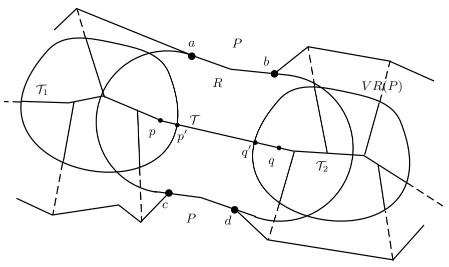


Fig. 4. A tree $\mathcal{T}$ of $VD(P)$ intersects $VR(P)$ at two connected subtrees $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively; $p \in \mathcal{T}_1$ and $q \in \mathcal{T}_2$, and p' and q' are two mixed vertices on the path pq; polygonal paths ab and cd are parts of P; circular arcs ac and bd are on the boundary of the spanning disks at p' and q' respectively; ab,ac,cd, bd form the boundary of a closed region R

RECURSIVE-FPVD
Input: A set of disjoint polygons $\mathcal{A}$
Output: FPVD of $\mathcal{A}$

```
1. If $\mathcal{A}$ is a singleton, i.e., $k = 1$, then the output is the (regular) Voronoi diagram of $\mathcal{A}$
2. Properly divide $\mathcal{A}$ into two sets $\mathcal{A}_1$ and $\mathcal{A}_2$
3. Recursively solve the two subproblems for $\mathcal{A}_1$ and $\mathcal{A}_2$
4. Merge the two FPVD's for $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively
```

The Voronoi diagram of disjoint segments in step 1 can be constructed in $O(n \log n)$ time. The proper division operation in step 2 can be done in $O(k \log k)$ time by sorting polygons by their sizes. Thus the remaining problem is how to merge the solutions of the two subproblems in the last step. It is shown in paper [3] that merging can be done in $O(n \log^2 n)$ time by using parametric searching technique. Below we present a more efficient algorithm that merges the two subproblems in $O(n \log n)$ time.

To merge the two FPVD's, we first determine those vertices in FPVD($\mathcal{A}_1$) and FPVD($\mathcal{A}_2$) which are vertices of the FPVD after the merge. It is known that the complexity of FPVD is $O(n)$ [3]. Thus an efficient point location algorithm is required. We use the point location algorithm in [5], which constructs the point location data structure of FPVD($\mathcal{A}$) in $O(n \log n)$ time and answers a query in $O(\log n)$ time. Thus, we first pre-compute the point location data structures for FPVD($\mathcal{A}_1$) and FPVD($\mathcal{A}_2$) respectively. The total running time for this pre-computation is $O(n \log n)$. Then for each vertex p of FPVD($\mathcal{A}_1$), find the Voronoi region of FPVD($\mathcal{A}_2$) that contains the vertex in $O(\log n)$ time by a query operation. Then calculate the distance from p to the farthest polygon in $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively (this can be done in $O(1)$ time). Let d_1 and d_2 be the distances. If $d_1 \geq d_2$, then p is a vertex of FPVD($\mathcal{A}$). Similarly, if p is a vertex of FPVD($\mathcal{A}_2$), query p on the point location data structure of FPVD($\mathcal{A}_1$) and calculate the largest distance d_1 and d_2 from p to polygons in $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. If

$d_1 \leq d_2$, then p is a vertex of FPVD($\mathcal{A}$). Since we have at most $O(n)$ vertices in total, the total time for query is $O(n \log n)$. Thus, it takes $O(n \log n)$ time to determine if the existing vertices in the two subproblems are vertices in the merged problem, and we have the following lemma.

Lemma 4. *The point location data structures for FPVD($\mathcal{A}_1$) and FPVD($\mathcal{A}_2$) can be pre-computed in $O(n \log n)$ time. For any point p in the plane, one can determine the farthest polygon P in $\mathcal{A}$ and the closest site s (open segment or endpoint) of that polygon in $O(\log n)$ time, i.e., one can find $P \in \mathcal{A}$ and $s \in P$ such that $p \in VR(s)$ in $O(\log n)$ time.*

By Lemma 3, we know that each tree in $VD(P)$ for any input polygon $P \in \mathcal{A}_1$(or $\mathcal{A}_2$) intersects $\mathcal{A}_1$ (or $\mathcal{A}_2$) at one single subtree or an empty tree. Let $\mathcal{T}$ be the subtree. Then, after the merge, $\mathcal{T} \cap FPVD(\mathcal{A})$ is also a connected subtree of $\mathcal{T}$. Let $\mathcal{T}' = \mathcal{T} \cap FPVD(\mathcal{A})$, and $\mathcal{I}(\mathcal{T}) = \{v \in \mathcal{T} | v \texttt{ is a node of } \mathcal{T} \texttt{ and } v \in \mathcal{T}'\}$. By Lemma 4, the set $\mathcal{I}$ of all trees of both FPVD($\mathcal{A}_1$) and FPVD($\mathcal{A}_2$) can be determined in $O(n \log n)$ time as the total number of vertices to query is $O(n)$. There are three cases of $\mathcal{I}(\mathcal{T})$ to consider.

1. All vertices of $\mathcal{T}$ are in $\mathcal{I}(\mathcal{T})$;
2. Some vertices of $\mathcal{T}$ are in $\mathcal{I}(\mathcal{T})$;
3. $\mathcal{I}(\mathcal{T})$ is empty

For case 1, $\mathcal{T}$ is intact after the merge, but the Voronoi region containing $\mathcal{T}$ can be shrunk after the merge due to the appearance of new pure vertices. For case 2 and 3, both the Voronoi region containing $\mathcal{T}$ and $\mathcal{T}$ itself are shrunk after the merge. Therefore the pure arcs that separate the shrunk Voronoi region and a Voronoi region of $\mathcal{A}_2$ have to be computed in order to correctly construct the FPVD. Thus we need to find the pure arcs that separate a Voronoi region of FPVD($\mathcal{A}_1$) and that of FPVD($\mathcal{A}_2$). It is essential to find the mixed vertices whose spanning disks touch sites in both $\mathcal{A}_1$ and $\mathcal{A}_2$. In fact, as we will show in Lemma 9, once such mixed vertices are all found, the merging problem of FPVD($\mathcal{A}$) can be trivially solved. In particular, for case 3, $\mathcal{T}'$ is not necessarily empty after the merge. $\mathcal{T}'$ can consist of a single arc which is partial arc of $\mathcal{T}$.

First, we present an algorithm to solve the following sub-problem called *pure arc tracing problem*: Given Voronoi cells C_1 of FPVD($\mathcal{A}_1$) and C_2 of FPVD($\mathcal{A}_2$) with C_1 being the Voronoi region of a site s_1 and C_2 being the Voronoi region of s_2 in their respective FPVD, and a vertex p on the pure arc separating $VR(s_1)$ and $VR(s_2)$ after the merge, find the next vertex q on the boundary of $VR(s_1)$ in a counter-clockwise order (see Figure 5).

In order to efficiently trace the arc, we need to further divide each Voronoi cell into smaller regions. Let C be a Voronoi cell and let s be the site of C. Then we have two cases to consider: s is an open line segment, and s is an endpoint. By Lemmas 1 and 2, $C = VR_i(s) \setminus \cup_{j \neq i} VR_{ij}(s)$, where $VR_i(s)$ and $VR_{ij}(s)$ have star or weakly star shape property. Therefore, we have the following lemma.

Lemma 5. *Let p be a point in C and q be the closest point of s. Then for any point p' on the segment pq such that $p' \in C$, segment pp' lies in C.*

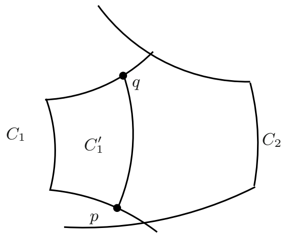


Fig. 5. The pure arc tracing problem. C_1' is the new Voronoi region of s_1 after merging.

We first subdivide a Voronoi cell C of either FPVD($\mathcal{A}_1$) or FPVD($\mathcal{A}_2$) as in Figure 6. We have two cases to consider. Case 1: if s is a line segment, we draw orthogonal lines from each vertex on the boundary to s. By Lemma 5, we know that each line intersects C at a connected line segment. Case 2: if s is a point, we draw lines connecting each vertex on the boundary and s. Similarly, the intersection is also a connected line segment. We call the intersection segments *doors*. The doors can be ordered along s (if s is a line segment) or circularly (if s is a point) around s. The main idea of our pure arc tracing algorithm is to do a binary search on the sorted list of doors of the cell and determine if the door and the arc intersect. Below are the main steps of the algorithm.

ARC-TRACING ALGORITHM

Input: A vertex p of the merged FPVD, two cells C_1 and C_2 containing p, and the corresponding sites s_1 and s_2, where $s_1 \in \mathcal{A}_1$ and $s_2 \in \mathcal{A}_2$;
Output: The next vertex q of the merged FPVD;

```
1. Subdivide $C_1$ and generate the sorted list of doors $D^1_1, D^1_2, \ldots, D^1_{j_1}$
2. Subdivide $C_2$ and generate the sorted list of doors $D^2_1, D^2_2, \ldots, D^2_{j_2}$
3. Compute the bisector $pp'$ of $s_1$ and $s_2$; W.L.G., assume $p > p'$ under the
   predicates for doors of both $C_1$ and $C_2$
4. If $p'$ is in $C_1$, go to step 12
5. Let $\lambda = 0, \mu = j_1 + 1$ // binary search on the doors of $C_1$ until $\mu - \lambda \leq 1$
6. $\zeta$ is the median of $\lambda, \lambda + 1, \ldots, \mu$
7. Compute the intersection point $p''$ between $pp'$ and the line extension of $D^1_\zeta$
8. If $p''$ does not exist, let $\lambda = \zeta$ or $\mu = \zeta$ so that $pp'$ and $D^1_\lambda$ lie on the same
   side of $D^1_\mu$
9. If $p''$ exists and lies in $C_1$, then $\mu = \zeta$
10. If $p''$ exists and lies outside of $C_1$, then $\lambda = \zeta$
11. Repeat step $6 - 10$ until $\mu - \lambda \leq 1$, then compute the intersection point $q$
    between $D^1_\lambda$ and $D^1_\mu$
12. If $q$ is in $C_2$, return $q$; otherwise, apply the same binary search procedure(step
    $5 - 11$) to $C_2$
13. Return $q$
```

Lemma 6. *The bisector pp' intersects the boundary of C_1 at most one point.*

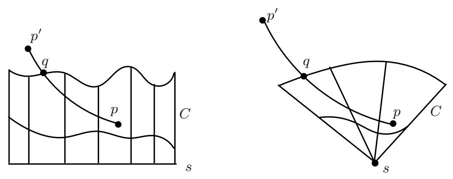


Fig. 6. Subdivision of a cell. The left figure is for the case that the site s is a line segment and the right one is for the case that s is a point.

Proof. Suppose pp' intersects the boundary of C_1 at two points. Then there are two disconnected Voronoi cells for s, which indicates that the tree associated with s intersects FPVD at two separate subtrees. This contradicts Lemma 3. $\square$

Lemma 7. *The doors of all cells in FPVD($\mathcal{A}_1$) and FPVD($\mathcal{A}_2$) can be computed in $O(n)$ time. If the doors are pre-computed and sorted, then algorithm ARC-TRACING can correctly find the next vertex in the merged FPVD in $O(\log{(n_1+n_2)})$ time, where n_1 and n_2 are the complexity of C_1 and C_2 respectively.*

Proof. The doors can be computed by traversing all vertices of the two FPVD's. For each vertex, draw an orthogonal line to s if s is a line segment or a line connecting the vertex and s if s is a point. Since the total complexity of the two FPVD's is $O(n)$, the running time for computing the doors is $O(n)$. The correctness of the algorithm follows from Lemma 3. For the running time, we only consider step $3-13$ because the first 2 steps are pre-computed. Step 3 takes $O(1)$ time. Step 4 takes $O(\log n)$ time by a point location query. Computing p'' in step 7 and the intersection points in line 12 take $O(1)$ time. Step 11 takes $O(1)$ time as well because we already know p'' is on D^1_ζ. So the loop from step 5 to 12 takes $O(\log n)$. Thus, the overall running time is $O(\log n)$ time if steps 1 and 2 have been pre-computed. $\square$

In fact, the ARC-TRACING algorithm is also applicable to the case that p is a point on a pure arc. Furthermore, the pure arc does not need to be a new one, i.e., the one that separates a cell of $\mathcal{A}_1$ and another cell of $\mathcal{A}_2$. In this case, we can make a trivial modification at line 3 of the algorithm by replacing the bisector pp' with the pure arc itself (in this case, the next vertex that it finds may be a new pure vertex). Thus, we know that the ARC-TRACING algorithm also handles this case.

Lemma 8. *The set of new bounding-box vertices, denoted by $\mathcal{B}$, can be computed in $O(n \log n)$ time.*

Proof. We first construct the convex hull of $\mathcal{A}$ in $O(n \log n)$ time [4]. Let $CH(\mathcal{A})$ be the convex hull. It is known that the complexity of $CH(\mathcal{A})$ is $O(n)$. We walk along the boundary of the convex hull in a counter-clockwise order. For each edge, if the two endpoints belong to different subsets, then there is an infinite arc of FPVD($\mathcal{A}_1$) which bisects the two corresponding sites. The intersection point between the bisection curve and the bounding-box can be computed in $O(1)$ time. Thus the total running time for computing all new bounding-box vertices is $O(n \log n)$. □

Lemma 9. *Let $\mathcal{M}$ be the set of new mixed vertices (i.e., the vertices whose minimal spanning disks touch sites in both $\mathcal{A}_1$ and $\mathcal{A}_2$). FPVD($\mathcal{A}$) can be computed in $O(n \log n)$ time.*

Proof. By Lemma 3, we know that each tree in $VD(P)$ for any input polygon $P \in \mathcal{A}_1$ (or $\mathcal{A}_2$) intersects $\mathcal{A}_1$ (or $\mathcal{A}_2$) at one single subtree or an empty tree. Let $\mathcal{T}$ be the subtree. Then $\mathcal{T} \cap FPVD(\mathcal{A})$ is a connected subtree of $\mathcal{T}$. Since there exist at least two vertices that are either mixed or bounding-box vertices for each Voronoi cell, we can apply the ARC-TRACING algorithm to compute all Voronoi cells of FPVD($\mathcal{A}$) (see Figure 7), starting from a mixed or bounding-box vertex of each cell. Since the existing vertices can be computed in $O(n \log n)$ time by querying every vertex on both point location data structures, by Lemmas 7 and 8, we know that the total running time for computing FPVD($\mathcal{A}$) is $O(n \log n)$ time. □

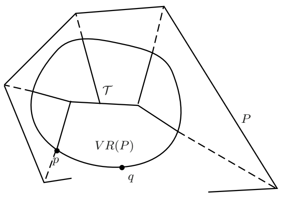


Fig. 7. The construction of FPVD($\mathcal{A}$) by arc-tracing from a known vertex p to a new vertex q

In fact, we need only to compute one mixed vertex(or a bounding-box vertex if there exists one) for each tree in FPVD($\mathcal{A}_1$) and FPVD($\mathcal{A}_2$). Then we can use the ARC-TRACING algorithm to compute each Voronoi cell of the merged FPVD, starting from the mixed or bounding-box vertex. We say that the tree is *trimmed* if some portion of the tree is discarded. A tree is *totally trimmed* if there is no new mixed vertex left. However, this observation does not improve the asymptotic running time in the worst case.

Now the remaining problem is how to efficiently compute the set of new mixed vertices $\mathcal{M}$. Our idea is to compute $\mathcal{M}$, trim the trees, and construct Voronoi cells of the merged FPVD simultaneously. We first introduce an important lemma.

Lemma 10. *Let $\mathcal{T}$ be a tree of FPVD($\mathcal{A}_1$) not totally trimmed, and $\mathcal{T}'$ be its totally trimmed tree. Let pq be an arc of $\mathcal{T}$ with p contained in a Voronoi cell C_1 of FPVD($\mathcal{A}_2$), s_1 and s_2 be two sites bisected by pq, and s_3 be the closest site of the farthest polygon P ($P \in \mathcal{A}_2$) of any point in C_1. Then there exists a mixed vertex p' of pq located in C_1 if and only if p' is the center of the disk touching s_1, s_2 and s_3 (see Figure 8).*

Proof. Let S_1 and S_2 be the corresponding Voronoi facets of C_1 and C_2 respectively in the Voronoi surface as shown in Figure 9. If ab intersects Voronoi surface S_1 at c, then $d_{i_1}(p') = d_{i_2}(p') = d_{i_3}(p')$, where $s_j \in P_{i_j}$ for $j = 1, 2, 3$. By the definition of d_i for Voronoi surfaces, we know $\rho(p', s_1) = \rho(p', s_2) = \rho(p', s_3)$, where ρ is the Euclidean distance. Thus p' is the center of the disk touching s_1, s_2 and s_3. Furthermore, p' is a mixed vertex because the angle between a facet and the plane is always 45^o. □

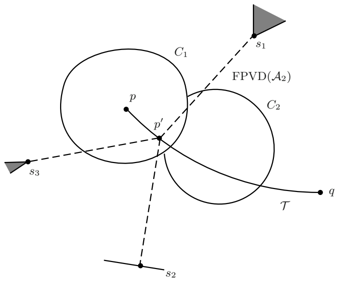


Fig. 8. Trimming tree $\mathcal{T}$ of FPVD($\mathcal{A}_1$). C_1 and C_2 are two Voronoi cells of FPVD($\mathcal{A}_2$); pq is an edge of $\mathcal{T}$; $p \in C_1$.

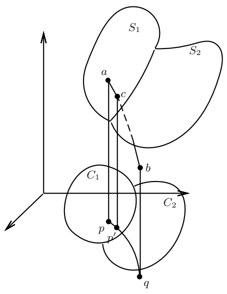


Fig. 9. Voronoi facets S_1 and S_2; C_1 and C_2 are the orthogonal projections of S_1 and S_2 respectively; p, q, p' are the orthogonal projections of a, b, c respectively

By symmetry, we can replace $\mathcal{A}_1$ with $\mathcal{A}_2$ in Lemma 10 and obtain a similar result for $\mathcal{A}_2$. To compute $\mathcal{M}$, for each tree $\mathcal{T}$, we totally trim $\mathcal{T}$ and generate all the mixed vertices by ARC-TRACING algorithm (note that this algorithm implicitly constructs the merged FPVD). We first roughly trim each tree $\mathcal{T}$ as follows. If $\mathcal{I}(\mathcal{T})$ is not empty, discard all edges whose both endpoints are not in $\mathcal{I}(\mathcal{T})$. If $\mathcal{I}(\mathcal{T})$ is empty, by Lemma 3, we know that $\mathcal{T}'$ is empty or consists of a single partial arc of $\mathcal{T}$. Furthermore, we can efficiently find the arc of $\mathcal{T}$ that contains $\mathcal{T}'$. In fact, for each tree $\mathcal{T}$, we can find a candidate arc that contains $\mathcal{T}'$ in $O(m \log n)$ time, where m is the complexity of $\mathcal{T}$ [3]. With this, we can trim

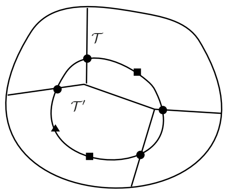


Fig. 10. Tree trimming and arc-tracing. The Voronoi region shrinks after the merge. $\mathcal{T}$ is the original tree in FPVD($\mathcal{A}_1$) and $\mathcal{T}'$ is the tree after the trimming. The dots are mixed vertices, the squares are pure vertices, and the triangle is the starting point of the ARC-TRACING algorithm.

the tree to a single candidate arc. The rough trimming process takes $O(n \log n)$ time. Now, we consider 4 cases to complete the trimming process for each tree and compute $\mathcal{M}$ (see Figure 10).

– Case 1: For each bounding-box vertex in $\mathcal{B}$, apply ARC-TRACING to totally trim the two trees contained by the two adjacent Voronoi regions. The new leaf nodes are added to $\mathcal{M}$;
– Case 2: For each tree $\mathcal{T}$ that is not trimmed in case 1, if there exists a mixed vertex in $\mathcal{I}(\mathcal{T})$, then use ARC-TRACING to totally trim $\mathcal{T}$. The resulting tree is denoted by $\mathcal{T}'$. The new leaf nodes are added to $\mathcal{M}$;
– Case 3: For each tree $\mathcal{T}$ that is not trimmed in case 1, if there is no mixed vertex in $\mathcal{I}(\mathcal{T})$. We first find one edge pq such that p is a medial axis vertex in $\mathcal{I}(\mathcal{T})$ and q is not in $\mathcal{I}(\mathcal{T})$. Then, as in Lemma 9, we find the center p' of the disk touching s_1, s_2, and s_3 in $O(1)$ time. Then determine if the center is on pq and in C_1. If it is, then starting from p', we completely trim $\mathcal{T}$ and add the newly-discovered mixed vertices into $\mathcal{M}$. Otherwise, we find the next neighboring Voronoi cell of C_1 along the trace of pq. Repeat the above procedure until finding the mixed vertex on pq.
– Case 4: For each tree $\mathcal{T}$ that is not trimmed in case 1, if $\mathcal{I}(\mathcal{T}) = \emptyset$, then $\mathcal{T}$ consists of one arc. Similar to case 3, but we may not have a mixed vertex on pq. To guarantee there is a mixed vertex on pq, let s be the site whose Voronoi cell contains p, then we can find the corresponding tree of s which also consists of one arc, say, $\alpha\beta$. Then there must exist a mixed vertex on $\alpha\beta$. Since we can find $\alpha\beta$ in $O(\log n)$ time, i.e., the time for point location query, we can assume there exists a mixed vertex on pq.

Once finding a point of some pure arc of the merged FPVD by one of the above 4 cases, we can compute $\mathcal{M}$ and construct the merged FPVD by ARC-TRACING algorithm. This is because all pure arcs are connected. Below are the main steps of the merging algorithm.

MERGE ALGORITHM
Input: FPVD of $\mathcal{A}_1$ and FPVD of $\mathcal{A}_2$, where $\mathcal{A}_1 \cup \mathcal{A}_2 = \mathcal{A}$
Output: FPVD of $\mathcal{A}$

```
1. Compute bounding-box vertices $\mathcal{B}$;
2. If $\mathcal{B}$ is not empty, for each bounding-box vertex $v$ of $\mathcal{B}$, compute $\mathcal{M}$ and construct the Voronoi cells of the merged FPVD that have $v$ on their boundary by ARC-TRACING algorithm; //Case 1;
3. For each vertex $v$ of FPVD($\mathcal{A}_1$) or FPVD($\mathcal{A}_2$), determine if $v$ is also a vertex of the merged FPVD; If yes, add $v$ to $\mathcal{I}$;
4. If $\mathcal{I}$ is not empty and there exists a mixed vertex $v$, then compute $\mathcal{M}$ and construct the rest of the merged FPVD by ARC-TRACING algorithm starting from $v$; // Case 2
5. If $\mathcal{I}$ is not empty but all vertices in $\mathcal{I}$ are medial axis vertices, then pick one vertex $v$ from $\mathcal{I}$ and find a tree $\mathcal{T}$ such that $\mathcal{I}(\mathcal{T})$ contains $v$; //Case 3
6. Traverse the vertices of $\mathcal{T}$ and find an arc $pq$ such that $p \in \mathcal{I}(\mathcal{T})$ but $q$ not in $\mathcal{I}(\mathcal{T})$;
7. Without loss of generality, $\mathcal{T}$ is a tree of FPVD($\mathcal{A}_1$); Let $s_1$ and $s_2$ be the two sites of $\mathcal{A}_1$ that $pq$ bisects;
8. Find the cell $C$ of FPVD($\mathcal{A}_2$) that contains $p$; Let $s_3$ be the site of $\mathcal{A}_2$ associated with $C$;
9. Compute the center $p'$ of the disk touching $s_1, s_2, s_3$;
10. If $p'$ is in $C$ and on $pq$, then $p'$ is a mixed vertex; Compute $\mathcal{M}$ and construct the rest of the merged FPVD by ARC-TRACING algorithm starting from $p'$;
11. Otherwise, $p$ is updated to be the intersection point between $pq$ and $C$, then repeat step $8 - 11$ until a mixed vertex is found in step 10;
12. If $\mathcal{I}$ is empty, roughly trim all trees of both FPVD's; //Case 4
13. Pick a trimmed tree that contains a mixed vertex; Since the tree consists of one arc, we can denote it by $pq$;
14. Find a mixed vertex by applying step $7-11$ and construct the merged FPVD;
```

Lemma 11. *The above algorithm takes $O(n \log n)$ time, where n is the complexity of the input.*

Proof. First, step 1 takes $O(n \log n)$ time by Lemma 8, and step 3 takes $O(n \log n)$ time by point location queries for at most $O(n)$ vertices. For steps $1, 10, 14$, each takes $O(n \log n)$ time by Lemma 7. This is because every ARC-TRACING discovers a new vertex and there are at most $O(n)$ vertices. Steps $8 - 11$ and 14 take $O(n \log n)$ time using the data structure in Figure 6. This is due to the fact that the total complexity of FPVD($\mathcal{A}_1$) and FPVD($\mathcal{A}_2$) is $O(n)$ and the intersection point can be computed in $O(\log n)$ time. It is easy to see that the rest of the algorithm is dominated by $O(n \log n)$. Thus the total running time is $O(n \log n)$. □

Theorem 2. *Given a set of disjoint polygons $\mathcal{A}$, the farthest-polygon Voronoi diagram can be computed in $O(n \log^2 n)$ time, where n is the total complexity of $\mathcal{A}$.*

Proof. Since we always have proper division of $\mathcal{A}$ and merging the solutions to the two subproblems FPVD($\mathcal{A}_1$) and FPVD($\mathcal{A}_2$) takes $O(n \log n)$ time by Lemma 11. Thus we have $T(n) \leq T(\frac{n}{4}) + T(\frac{3n}{4}) + O(n \log n)$, where $T(n)$ is the total time for computing FPVD(A). This solves to $T(n) = O(n \log^2 n)$, since the height of the recursive tree is at most $O(\log n)$. □

4 Conclusion

In this paper, we studied two Voronoi diagram problems, Farthest Colored Voronoi Diagram (FCVD) and Farthest-polygon Voronoi Diagram (FPVD). For the former, we present a simple output-sensitive algorithm with running time $O((kn + I)\alpha(H) \log n)$, where n is the complexity of the input, k is the number of colors, I is the number of intersection points, and H is the complexity of the FCVD. For the latter, we present an $O(n \log^2 n)$-time algorithm using divide-and-conquer paradigm and arc-tracing technique. Comparing to the best known algorithm, our algorithm does not rely on parametric search, is simpler, and improves the time complexity by a factor of $O(\log n)$. As an open problem, it would be interesting to design an $O(n \log n)$-time algorithm for FPVD.

References

1. Agarwal, K.P., Sharir, M.: Algorithmic techniques for geometric optimization (1995)
2. Aurenhammer, F., Drysdale, R.L.S., Krasser, H.: Farthest line segment voronoi diagrams. Inf. Process. Lett. 100(6), 220–225 (2006)
3. Cheong, O., Everett, H., Glisse, M., Gudmundsson, J., Hornus, S., Lazard, S., Lee, M., Na, H.-S.: Farthest-polygon voronoi diagrams. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 407–418. Springer, Heidelberg (2007)
4. de Berg, M., Van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational geometry: algorithms and applications, 2nd edn. (2000)
5. Edelsbrunner, H., Guibas, L.J., Stolfi, J.: Optimal point location in a monotone subdivision. SIAM J. Comput. 15, 317–340 (1986)
6. Huttenlocher, D.P., Kedem, K., Sharir, M.: The upper envelope of voronoi surfaces and its applications. In: SCG 1991: Proceedings of the seventh annual symposium on Computational geometry, pp. 194–203. ACM, New York (1991)
7. Nielsen, F., Yvinec, M.: An output sensitive convex hull algorithm for planar objects. International Journal of Computational Geometry and Applications 8, 39–65 (1995)
8. Yap, C.K.: An o(n logn) algorithm for the voronoi diagram of a set of simple curve segments. In: Discrete and computational geometry, vol. 2, pp. 365–393 (1987)

One-and-a-Half-Side Boundary Labeling

Chun-Cheng Lin[1,⋆], Sheung-Hung Poon[2,⋆⋆], Shigeo Takahashi[3], Hsiang-Yun Wu[3], and Hsu-Chun Yen[4,⋆⋆⋆]

[1] Dept. of Industrial Engineering and Management, National Chiao Tung University, Hsinchu 300, Taiwan
[2] Dept. of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan
[3] Dept. of Complexity Science and Engineering, The University of Tokyo, Kashiwa-city, Chiba 277-8561, Japan
[4] Dept. of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan

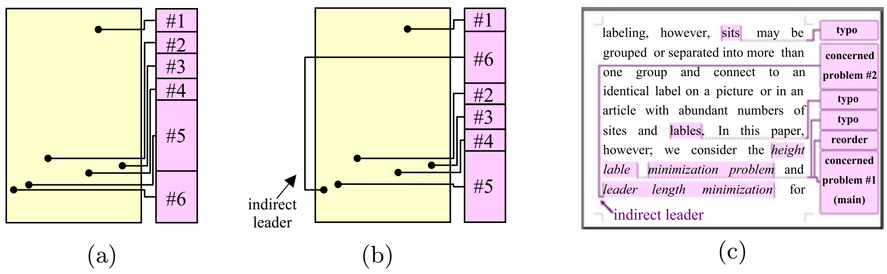
Abstract. In *boundary labeling*, each point site in a rectangular map is connected to a label outside the map by a *leader*, which may be a rectilinear or a straight-line segment. Among various types of leaders, the so-called type-*opo* leader consists of three segments (from the site to its associated label) that are orthogonal, then parallel, and then orthogonal to the side to which the label is attached. In this paper, we investigate the so-called *1.5-side boundary labeling*, in which, in addition to being connected to the right side of the map directly, type-*opo* leaders can be routed to the left side temporarily and then finally to the right side. It turns out that allowing type-*opo* leaders to utilize the left side of a map is beneficial in the sense that it produces a better labeling result in some cases. To understand this new version of boundary labeling better, we investigate from a computational complexity viewpoint the *total leader length minimization* problem as well as the *bend minimization* problem for variants of *1.5-side boundary labeling*, which are parameterized by the underlying label size (uniform vs. nonuniform) and port type (fixed-ratio, fixed-position, vs. sliding). For the case of nonuniform labels, the above two problems are intractable in general. We are able to devise pseudo-polynomial time solutions for such intractable problems, and also identify the role played by the number of distinct labels in the overall complexity. On the other hand, if labels are identical in size, both problems become solvable in polynomial time. We also characterize the cases for which utilizing the left side for routing type-*opo* leaders does not help.

Keywords: Map labeling, boundary labeling, complexity.

1 Introduction

In map labeling [5,10,11], the basic requirement for placing labels in a map is that all the labels should be pairwise disjoint. It is clear that such a requirement is

⋆ Corresponding author (E-mail: `cclin321@nctu.edu.tw`). Research supported in part by NSC 98-2218-E-151-004-MY3.
⋆⋆ Research supported in part by NSC 97-2221-E-007-054-MY3.
⋆⋆⋆ Research supported in part by NSC 97-2221-E-002-094-MY3.



labeling, however, sits may be grouped or separated into more than one group and connect to an identical label on a picture or in an article with abundant numbers of sites and lables. In this paper, however; we consider the *height lable minimization problem* and *leader length minimization* for

Fig. 1. (a) One-side boundary labeling with type-*opo* leaders. (b) 1.5-side boundary labeling with type-*opo* leaders. (c) Application to a word processing annotation system.

difficult to be achieved in the case where large labels are placed on dense points. To address this problem, Bekos et al. [3] proposed the so-called *boundary labeling*, in which all labels are attached to the boundary (four sides) of a rectangle R enclosing all sites, and each site is connected to a unique label by a *leader*, which may be a rectilinear or a straight-line segment. In such a setting, they assumed no two sites with the same x- or y- coordinates, and investigated how to place the labels and leaders in a drawing such that there are no crossings among leaders and either the total leader length or the total number of bends of leaders is minimized under a variety of constraints. Bekos et al. [2] investigated a similar problem for labeling polygonal sites under the framework of boundary labeling. Subsequently, Lin [7] used hyperleaders and multiple copies of the same label to propose algorithms for crossing-free multi-site-to-one-label boundary labeling [8], in which more than one site is allowed to be connected to a common label.

Boundary labeling [3,4,1] is characterized as *k-side labeling with type-t leaders* (where $k \in \{1, 2, 4\}$ and $t \in \{opo, po, s, do\}$) if the labels are allowed to attach to the k sides of the enclosing rectangle R by only type-t leaders. The parameter t specifies the way in which a leader is drawn to connect a site to a label. The *opo*, *po*, *s*, and *do* stand for *orthogonal-parallel-orthogonal*, *parallel-orthogonal*, *straight-line* and *diagonal-orthogonal* leader types, respectively. It is assumed that the parallel (i.e., 'p') segment associated with a type-*opo* leader lies in a *track routing area* sandwiched between R and the label stack. See Figure 1(a). In a recent article [9], the idea of the so-called *indirect* leaders was proposed for possibly shortening the total leader length in one-side boundary labeling. *Indirect* leaders utilize the left side of the map to route the parallel segments of type-*opo* leaders (while the original ones that do not utilize the left side are called *direct* leaders). See Figure 1(b) for an example. Such a new type of boundary labeling is called *1.5-side boundary labeling with type-opo leaders*, which is likely to find applications in, for example, text annotation for word processing softwares as [9] suggests (see Figure 1(c) for an illustrating example). In general, the track routing area is not wide, so too many direct leaders in this area would make it difficult to tell them apart (see Figure 1(a)). The introduction of indirect leaders may lead to fewer direct leaders in this area, where the routing of those direct leaders can be distinguished more easily (see Figure 1(b)).

In this paper, we consider, from a computational complexity viewpoint, the *total leader length minimization* (*TLLM*) problem and the *total bend minimization* (*TBM*) problem for variants of *1.5-side boundary labeling with type-opo leaders*, which are parameterized by their label and port types. A label can be of *uniform* or *nonuniform* size, and the port (i.e., the position where a leader touches a label) associated with a label is of type *fixed-ratio*, *fixed-position*, or *sliding*. It turns out that for nonuniform labels, both *TLLM* and *TBM* are intractable in general regardless of the port type. We are able to design a pseudo-polynomial time algorithm and a fixed-parameter algorithm for such intractable problems. Both *TLLM* and *TBM* become solvable in polynomial time if labels are of uniform size. Interestingly, we also show that for labels of uniform size and under either sliding or fixed-ratio port type, indirect leaders do not help as far as minimizing the total leader length is concerned.

2 Preliminaries

2.1 The Models for 1.5-Side Boundary Labeling

In 1.5-side boundary labeling, we assume that sites are points of zero size located on the plane, and only type-*opo* leaders are used, no matter whether they are direct or indirect. Following Bekos et al.'s convention [2], various models for 1.5-side boundary labeling can be differentiated according to a triple (*LabelSize*, *LabelPort*, *Objective*), where:

LabelSize: Each label l_i is associated with a height h_i and a width w_i. As each leader is connected to the left side of a label box, w.l.o.g., we assume that $\forall 1 \leq i, j \leq n, w_i = w_j$, where n is the number of labels. Labels are of *uniform* size if $\forall 1 \leq i, j \leq n, h_i = h_j$; otherwise, of *nonuniform* size.

LabelPort: Depending on the location where a leader touches a label, consider the following three types:

- *Fixed-ratio port* (*FR* for short): there exists a constant $0 \leq \alpha \leq 1$, such that the i-th leader touches the point of height αh_i, from the bottom of the i-th label. In [8], α is assumed to be $\frac{1}{2}$, i.e., each leader touches the middle of the corresponding label.
- *Fixed-position port* (*FP* for short): The i-th label is associated with a predefined constant $0 \leq \alpha_i \leq 1$ such that the i-th leader touches the point of height $\alpha_i h_i$, from the bottom of the i-th label.
- *Sliding port*: As the name suggests, the contact point of a leader can *slide* along the corresponding label edge.

Objective: Find a legal label placement such that the total leader length is minimum (*TLLM*), or the total number of bends is minimum (*TBM*).

Before proceeding further, we first show three examples as depicted in Figure 2, which suggests that indirect leaders really help.

Our main results are given in Table 1, where the uniform-label cases can be solved in polynomial time, while the nonuniform-label cases are NP-complete.

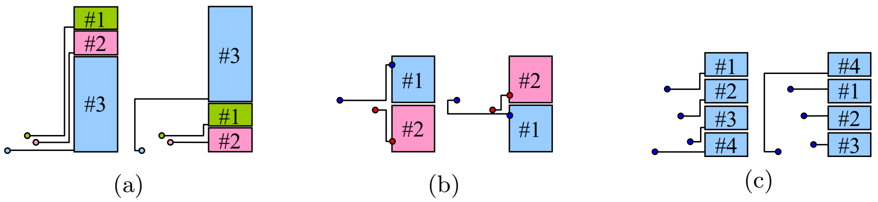


Fig. 2. (a) *TLLM* with nonuniform labels; (b) *TLLM* with uniform labels and *FP* ports; (c) *TBM* with uniform labels

Table 1. Time complexity for a variety of 1.5-side boundary labeling models. *FR* denotes fixed-ratio port, and *FP* denotes fixed-position port.

(*LabelSize*,	*LabelPort*,	*Objective*)	time	reference
(*uniform*,	*FR/sliding*,	*TLLM*)	$O(n \log n)$	Thm 1
(*uniform*,	*FP*,	*TLLM*)	$O(n^5)$	Thm 2
(*uniform*,	*FR/FP/sliding*,	*TBM*)	$O(n^5)$	Thm 2
(*nonuniform*,	*FR/FP/sliding*,	*TLLM*)	NP-complete*	Thm 3
(*nonuniform*,	*FR/FP/sliding*,	*TBM*)	NP-complete*	Thm 4

* An $O(n^4h)$ pseudo-polynomial time algorithm is developed in Theorem 5, where h is the height of the map. Furthermore, if labels are of k different heights, an $O(n^{k+4})$ time algorithm is available as Theorem 6 shows.

2.2 Problem Setting

We consider the following labeling problem. Given a rectangular area R of height h and width w whose left lower corner resides at the origin of the x-y plane (i.e., $R = [0, w] \times [0, h]$), and a set of n points (called *sites*) $p_i = (x_i, y_i), 1 \leq i \leq n$, located inside R (i.e., $0 \leq x_i \leq w, 0 \leq y_i \leq h, 1 \leq i \leq n$), each of which is associated with a rectangular label l_i of width w_i and height h_i, the *one-and-a-half-side* (1.5-side, for short) *boundary labeling* problem is to place the labels along one side of the boundary of R, and connect p_i to l_i, $1 \leq i \leq n$ using *rectilinear leaders* that are either direct or indirect (or leaders, for short) so that a certain criterion is met. As illustrated in Figure 1, a *rectilinear leader* consists of horizontal and/or vertical line segments connecting a site to its corresponding label. We assume that h, w, x_i, y_i, h_i, w_i, $1 \leq i \leq n$, are all positive integers. We further require that a leader has at most two bends, which occur only in one of the two *track routing areas* denoted as A_{left} and A_{right} (see Figure 3). A leader bending in the area A_{right} (resp., A_{left}) is called a *direct* (resp., *indirect*) leader. Throughout the rest of this paper, we assume that there are no two sites with the same x- or y- coordinate, and sites are labeled as $p_1, p_2, \cdots, p_n$ in the increasing order of their y-coordinates.

We assume that $\sum_{i=1}^{n} h_i = h$, i.e., the label heights sum up to the height of R. In this case, the y-coordinate of the top of the j-th label is $\sum_{i=1}^{j} h'_i$, where h'_i is the height of the i-th label from the bottom of the label stack. Note

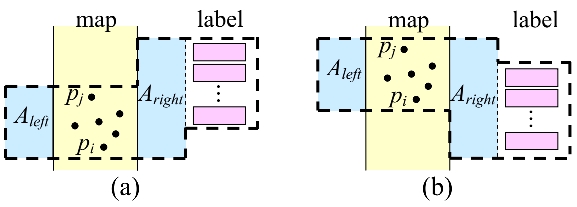


Fig. 3. Illustration of two cases for $G(i,j)$

that since we allow indirect leaders, label l_j need not be the j-th label. For the criteria considered in this paper, we may further assume that labels are of uniform width (i.e., $w_i = w_j, \forall 1 \le i,j \le n$) with no loss of generality. Consider a region $G(i,j)$ induced by a subset of sites $p_i, \cdots, p_j$ and their corresponding labels $l_i, \cdots, l_j, 1 \le i < j \le n$ as shown in Figure 3. Note that the left track routing area is of height less than $(y_{j+1} - y_{i-1})$, and the height of the right track routing area depends on the relative positions of the sites $p_i, \cdots, p_j$ and their labels. A leader connecting a site p_k to its corresponding label l_k (where $1 \le k \le j$) is *legal* with respect to $G(i,j)$ if it resides entirely in $G(i,j)$.

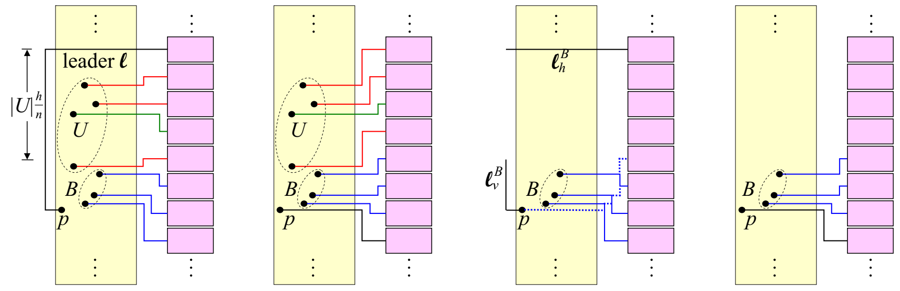
3 Uniform-Label Cases

Consider the uniform-label case, i.e., $h_i = \frac{h}{n}, \forall 1 \le i \le n$. As Example 2(b) indicates, using indirect leaders may result in a shorter total leader length in some cases, provided that leaders are connected to fixed-position ports. In contrast, if the ports are fixed-ratio (with respect to all labels) or can slide along boundaries of labels, the following result shows indirect leaders to be unnecessary:

Lemma 1. *In the case of uniform labels, direct leaders are sufficient to achieve optimal solutions with respect to TLLM under either the fixed-ratio port or the sliding port model.*

Proof (Sketch). In the following, we only show the fixed-ratio port model, because the proof for the sliding port model is similar. It suffices to prove that the total leader length of any labeling L_i with i indirect leaders is no shorter than that of the labeling δ with only direct leaders. Let $\phi(L)$ denote the total leader length of labeling L. The basic idea of our proof is to find labelings L_{i-1}, L_{i-2}, $\ldots$, L_1 such that $\phi(L_i) \ge \phi(L_{i-1}) \ge \cdots \ge \phi(L_1) \ge \phi(\delta)$, where L_j denotes a labeling with j indirect leaders for $1 \le j < i$.

Consider the inner-most indirect leader (i.e., the indirect type-*opo* leader has the rightmost parallel segment) in labeling L_i, which is denoted by ℓ, and the site connected with ℓ is denoted by p (see Figure 4(a)). Hence, all the sites wrapped by ℓ are connected only by direct leaders, and they are divided into two groups according to the orientation of their associated direct leaders. Consider each of those wrapped sites from the bottom to the top, say site q. From site q to its corresponding label, if the vertical segment of the associated leader goes



(a) Labeling L_i. (b) Labeling L_{i-1}. (c) (d)

Fig. 4. The inner-most indirect leader in labeling (a) is replaced by a direct leader in labeling (b). (c) and (d) are respectively the labeling (a) and (b) where the total leader length of (b) for those sites in U are removed. Note that the leader segments that occur in (d) but not in (c) are represented by dotted-line segments in (c).

downward and all the sites located below q belong to group B, then $q \in B$; otherwise, $q \in U$ (see Figure 4(a)). That is, there exist sites with leaders going downward but not belonging to group B. Let $\Delta = \{p\} \cup U \cup B$.

If $\Delta = \{p\}$ only, then indirect leader ℓ can be changed to a direct one immediately, and the modified labeling is our required labeling L_{i-1}. Otherwise, we establish a labeling L_{i-1} with $(i-1)$ indirect leaders which is almost the same as labeling L_i except all the sites in Δ are connected by direct leaders (see Figure 4(b)). Let $d(p_j)$ (resp., $d'(p_j)$) denote the length of the leader connected to site p_j in L_i (resp., L_{i-1}), for any $j \in \{1, \cdots, n\}$. For any set of sites, let $d(A) = \sum_{p_j \in A} d(p_j)$, and define $d'(A)$ similarly. From Figures 4(a) and 4(b), it is observable that $d(U) + |U|\frac{h}{n} \geq d'(U)$. Since $d(p) > |U|\frac{h}{n}$ in labeling L_i (see Figure 4(a)), the lengths $d(U) + |U|\frac{h}{n}$ and $d'(U)$ can be removed in L_i and L_{i-1}, respectively. Figure 4(c) (resp., Figure 4(d)) is the labeling L_i in (a) (resp., L_{i-1} in (b)) removing the length $d(U)+|U|\frac{h}{n}$ (resp., $d'(U)$), i.e., the total leader length of L_i is decreased no less than that of L_{i-1}. Note that in Figure 4, the leader segments that occur in (d) but not in (c) are represented by dotted-line segments in (c). It is easy to see that the remaining length of the indirect leader ℓ in (c) can be used to compensate those dotted-line segments in (c), in which lengths ℓ_h^B and ℓ_v^B compensate dotted horizontal and vertical segments, respectively. Even so, there still exists a nonnegative remaining length of the indirect leader ℓ in (c). Hence, $\phi(L_i) \geq \phi(L_{i-1})$, as required.

Like the above, we can construct L_{j-2} according to L_{j-1} such that $\phi(L_{j-1}) \geq \phi(L_{j-2})$, and so forth. Finally, we have $\phi(L_i) \geq \phi(L_{i-1}) \geq \cdots \geq \phi(L_1) \geq \phi(\delta)$, as required. □

According to Lemma 1 above and [3] (the boundary labeling with direct leaders can be found in $O(n \log n)$ time), we have the following theorem.

Theorem 1. *The* 1.5*-side boundary labeling of the model* (*uniform*, *FR/sliding*, *TLLM*) *can be found in* $O(n \log n)$ *time.*

Next, we use a dynamic programming strategy to solve both *TLLM*[1] and *TBM* for 1.5-side boundary labeling with type-*opo* leaders in $O(n^5)$ time, regardless of the underlying port model. Note that according to Theorem 1, *TLLM* can be solved in $O(n \log n)$ time under the *FR/sliding* port model.

Theorem 2. *With respect to opo-type* 1.5*-side boundary labeling, both TLLM and TBM can be solved in* $O(n^5)$ *time when labels are of uniform height, regardless of the underlying port model.*

Proof. (Sketch) In what follows, we only consider *TLLM*; the *TBM* case is similar.

The proof is based on the strategy of dynamic programming. Without loss of generality, we assign indices j $(1 \leq j \leq n)$ to labels in the label stack in the increasing order of their y-coordinates. Note that due to the nature of 1.5-side boundary labeling, label l_i (i.e., the one associated with site p_i) need not be the i-th label in the label stack. It is easy to see that the y-coordinates of the top and bottom edges of the i-th label from the bottom of the label stack are $i \times \frac{h}{n}$ and $(i-1) \times \frac{h}{n}$, respectively. In the case of using fixed-ratio ports, we define $\gamma(p_i, j)$ (resp., $\ell(p_i, j)$) to be the length of a direct (resp., indirect) type-*opo* leader from site p_i to the fixed-position port of the j-th label in the label stack. In the case of using sliding ports, $\gamma(p_i, j)$ and $\ell(p_i, j)$ are defined similarly except that a leader connects p_i to the closest point (as opposed to the fixed-position point) on the boundary of the j-th label.

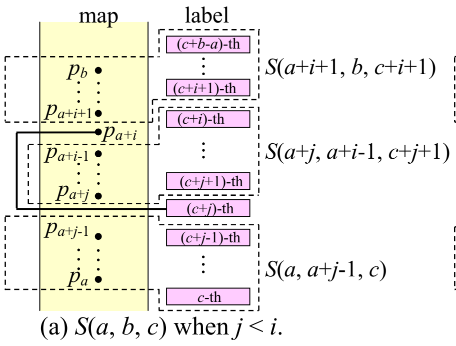
Let $S(a, b, c)$ (where $1 \leq a \leq b \leq n, 1 \leq c \leq n$) denote the minimal total leader length where sites $p_a, \cdots, p_b$ $(1 \leq a \leq b \leq n)$ are connected to the labels of the label stack that are consecutive in the bottom to top order starting from the c-th one up to the $(c-(b-a))$-th one using only legal leaders. We define $R_{a,b}$ to be the rectangle $[0, w] \times [y_{a-1}, y_{b+1}]$. Note that $R_{a,b}$ defines the left track routing area through which any indirect leader from site $p_i, a \leq i \leq b$ can be routed. In other words, it is illegal for p_i to use an indirect leader to connect to a label whose y-coordinate is above y_{b+1} or below y_{a-1}.

Our dynamic programming formula for $S(a, b, c)$ is as follows (see Figure 5):

$$\begin{aligned}
\min\{&\sum_{i=0}^{b-a} \gamma(p_{a+i}, c+i),\\
&\min_{i,j \in \{0,\cdots,b-a\}, j<i} \{\ell(p_{a+i}, c+j) + S(a, a+j-1, c)\\
&\qquad +S(a+j, a+i-1, c+j+1) + S(a+i+1, b, c+i+1)\},\\
&\min_{i,j \in \{0,\cdots,b-a\}, j>i} \{\ell(p_{a+i}, c+j) + S(a, a+i-1, c)\\
&\qquad +S(a+i+1, a+j, c+i) + S(a+j+1, b, c+j+1)\}\}
\end{aligned}$$

In view of the above, it is reasonably easy to see that the minimum total leader length equals $S(1, n, 1)$.

[1] A dynamic programming formulation was originally given for a simpler version of *TLLM* in [9].



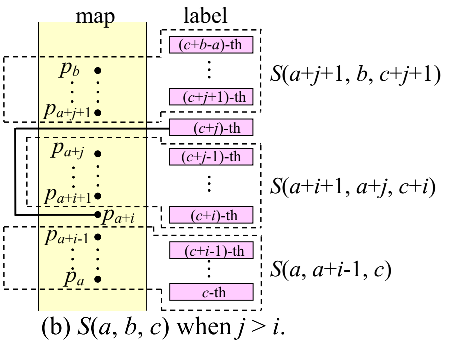


Fig. 5. Illustration of a subproblem

As for the complexity of the algorithm, we construct n tables $T_1, \cdots, T_n$ where table T_i has $(n-i+1) \times (n-i+1)$ entries for $i \in \{1, \cdots, n\}$; the entry (j,k) in table T_i records the minimal total leader length when sites $p_j, p_{j+1}, \cdots, p_{j+i-1}$ are connected to the k-th, the $(k+1)$-th, $\cdots$, and the $(k+i-1)$-th labels in the label stack by direct and indirect type-*opo* leaders. By doing so, the solution of our problem can be found in the entry $T_n(1,1)$. In view of the dynamic programming formula, each entry of Table $T_i, 1 \leq i \leq n$, can be computed in $O(i^2)$ time. Hence, using a bottom up approach, $T_n(1,1)$ can be obtained in time $O(\sum_{i=1}^{n}(n-i+1)^2 i^2) = O(n^5)$. □

4 Nonuniform-Label Cases

4.1 NP-Hardness

In this subsection, we show that the 1.5-side boundary labeling for non-uniform labels is NP-complete for both *TLLM* and *TBM*.

Theorem 3. *It is NP-complete to find a* 1.5*-side boundary labeling of model* (*non- uniform*, *FR/FP/sliding*, *TLLM*).

Proof (Sketch). We only consider the *FR/FP*-port model here. The NP-complete proof for the sliding-port model can be shown along a similar line, and therefore, is omitted here. To see that the problem is in NP, since we assume that h, w, x_i, y_i, h_i, w_i, $1 \leq i \leq n$, are all positive integers. In order to show the NP-hardness of our problem, we obtain a linear-time reduction from a single-machine scheduling problem, called *total discrepancy problem* [6], to our problem. On one machine, we plan to arrange the schedule for the non-preemptive execution of a set J of $2n+1$ jobs $J_0, J_1, \ldots, J_{2n}$. Each job J_i has an execution time length $l_i \in \mathbb{Z}^+$ such that $l_0 < l_1 < \ldots < l_{2n}$. For a planned schedule σ, the actual execution midtime for job J_i is denoted by $m_i(\sigma)$. Each job has a *preferred midtime*, which corresponds to the time at which we would like the first half of the job to be completed. We assume that all the jobs share a single preferred

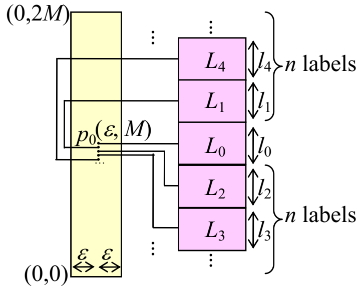


Fig. 6. Reduction in Theorem 3

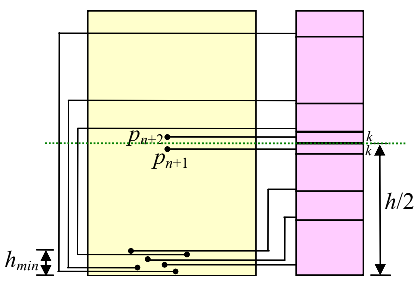


Fig. 7. Reduction in Theorem 4

midtime $M = \sum_{i=0}^{2n} l_i/2$. The *penalty* of job J_i for a schedule σ is defined as the absolute difference of its midtime to its preferred midtime, i.e., $|m_i(\sigma) - M|$ for $0 \le i \le 2n$. The *cost* incurred in a schedule σ is then defined to be the total penalties incurred by all jobs. The objective of the total discrepancy problem is to determine a schedule σ such that the total cost of the schedule, i.e., $\sum_{i=0}^{2n} |m_i(\sigma) - M|$, is minimized. Garey, Tarjan and Wilfong [6] showed the following properties for an optimal schedule σ_{opt} of the $2n+1$ jobs $J_0, J_1, \cdots, J_{2n}$, but the decision problem whether such a schedule exists is NP-complete.

1. σ_{opt} does not have any gaps between the jobs.
2. The midtime of the shortest job J_0 is M, i.e., $m_0(\sigma_{opt}) = M$.
3. Jobs $J_1, J_2, \ldots, J_{2n}$ are divided into two groups, $A(\sigma_{opt}) = \{J_i : m_i(\sigma_{opt}) < M\}$ and $B(\sigma_{opt}) = \{J_i : m_i(\sigma_{opt}) > M\}$, such that $|A(\sigma_{opt})| = |B(\sigma_{opt})| = n$.
4. Suppose the sequence of the jobs in schedule σ_{opt} is $A_n, A_{n-1}, \ldots, A_1, J_0, B_1, B_2, \ldots, B_n$. Then $\{A_i, B_i\} = \{J_{2i-1}, J_{2i}\}$.
5. The optimal cost is equal to $\sum_{i=1}^{n}(l_{2i} + l_{2i-1})(n - i + 1/2) + nl_0$.

From this version of the scheduling problem, we show how it can be reduced to our problem in the following. We set ϵ to be a very small constant value less than, say for example $\min\{l_0, \min_{i=0}^{2n-1}(l_{i+1} - l_i)/(100n^3)\}$. We set the map area to be $[0, 2\epsilon] \times [0, 2M]$ (see Figure 6).

We put all point sites in the map area along the vertical line $x = \epsilon$. For job $J_i, i = 0, 1, \cdots, 2n$, we introduce its corresponding point site p_i placing at location $(\epsilon, M - i\epsilon)$ such that the leader for p_i connects to the middle position of label L_i with height l_i (see Figure 6). Moreover, we set $k = \sum_{i=1}^{n}(l_{2i} + l_{2i-1})(n - i + 1/2) + nl_0$. Since ϵ is very small compared to any label height, we nearly can treat the locations of point sites p_i to be exactly lying at (ϵ, M). Under such a scenario, it can be shown that there is a scheduling with cost at most k if and only if there is a legal labeling with total leader length at most $k + l_0/n$. $\square$

Theorem 4. *It is NP-complete to find a* 1.5*-side boundary labeling of the model (non- uniform, FR/FP/sliding, TBM).*

Proof (Sketch). We only discuss the sliding port case, because the fixed-ratio port case is similar. It is obvious to see that the problem is in NP. Hence, it suffices to show that the problem is NP-hard. The proof of the NP-completeness is based on the reduction from the following *subset sum* problem: Given $A = \{a_1, a_2, \cdots, a_n\}$ and a number B, the objective of the problem is to find a subset $A' \subset A$ such that the sum of the elements in A' is exactly B.

Given a set $A = \{a_1, \cdots, a_n\}$ and a number B, we construct a map instance as follows. There are $n + 2$ sites p_1, p_2, ..., p_{n+2} in the map instance. The height of the label connected with site p_i is denoted by h_i, in which $h_{min} < \min\{h_1, \cdots, h_n\}$ and $h_{min} > n$. Let $\{h_1, \cdots, h_n\} = \{a_1, \cdots, a_n\}$, $h_{n+1} = h_{n+2} = k \leq h_{min}$, and $a_1 + \cdots + a_n = 2B$. Note that the y-coordinate of the bottom side of the map is zero. For each $i \in \{1, \cdots, n\}$, $y(p_i) \leq h_{min}$, $y(p_{n+1}) = h/2 - \epsilon$, and $y(p_{n+2}) = h/2 + \epsilon$, where h is the height of the map and $\epsilon < h_{min}$.

See also Figure 7. It is easy to see that the sum of the elements in A' is exactly $B = (a_1 + \cdots + a_n)/2$ if and only if the number of bends is $2n - 2$. □

4.2 Pseudo-polynomial Time Algorithm

An idea which parallels the one used in Theorem 2 is used to show the following:

Theorem 5. *With respect to* 1.5*-side boundary labeling, both TLLM and TBM can be solved in* $O(n^4 h)$ *time when labels are of nonuniform height and ports are either fixed-ratio, fixed-position, or sliding.*

Proof. We only show how to solve the *TLLM* problem, since the *TBM* problem can be solved similarly. Like in the proof of Theorem 2, our pseudo-polynomial time algorithm is again based on the strategy of dynamic programming.

In the case of using fixed-ratio or fixed-position ports, we define $\gamma(p_i, t)$ (resp., $\ell(p_i, t)$), $1 \leq i \leq n, 0 \leq t < t' \leq h$, to be the length of a direct (resp., indirect) type-*opo* leader from site p_i to the fixed port of l_i when the y-coordinate of the bottom (resp., top) of the label is t (resp., t') in the label stack, provided that $t' - t = h_i$; if $t' - t \neq h_i$, then $\gamma(p_i, t)$ (resp., $\ell(p_i, t)$) $= \infty$. Note that t' is calculated in the dynamic programming procedure. In the case of using sliding ports, $\gamma(p_i, t)$ and $\ell(p_i, t)$ are defined similarly except that a leader connects p_i to the closest point (as opposed to the fixed point) on the boundary of l_i. We let $S(a, b, t)$ (where $1 \leq a \leq b \leq n, 0 \leq t \leq h$) denote the minimal total leader length where sites $p_a, \cdots, p_b$ $(1 \leq a \leq b \leq n)$ are connected to their labels $l_a, \cdots, l_b$ which are placed in the label stack with t as the y-coordinate of the bottom of the lowest label. In other words, labels $l_a, \cdots, l_b$ occupy the area whose y-coordinate ranges from t to $t + \sum_{i=a}^{b} h_i$. We define $R_{a,b}$ to be the rectangle $[0, w] \times [y_{a-1}, y_{b+1}]$.

Our dynamic programming formula for $S(a, b, t)$ is as follows:

$$\min\{\sum_{i=0}^{b-a} \gamma(p_{a+i}, t + \sum_{j=0}^{i-1} h_{a+j}),$$

$$\min_{i,j\in\{0,\cdots,b-a\},j<i}\{\ell(p_{a+i},t+\sum_{l=0}^{j-1}h_{a+l})+S(a,a+j-1,t)$$

$$+S(a+j,a+i-1,t+(\sum_{l=0}^{j-1}h_{a+l})+h_{a+i})+S(a+i+1,b,t+\sum_{l=0}^{i}h_{a+l})\},$$

$$\min_{i,j\in\{0,\cdots,b-a\},j>i}\{\ell(p_{a+i},t+\sum_{l=0}^{j-1}h_{a+l})+S(a,a+i-1,t)$$

$$+S(a+i+1,a+j,t+\sum_{l=0}^{i-1}h_{a+l})+S(a+j+1,b,t+\sum_{l=0}^{j}h_{a+l})\}\}$$

The reason why the above dynamic programming formula correctly characterizes $S(a,b,t)$ is similar to that in the proof of Theorem 2. □

4.3 Fixed-Parameter Algorithm

We present a polynomial time algorithm when the number of different heights of labels, k, is a constant. First, we have the observation that the number of possible label positions is $O(n^k)$.

Lemma 2. *The number of possible label positions of each label is bounded by $O(n^k)$, if the given labels are of k different heights.*

Proof. We prove this lemma by induction on k. When $k=1$, it is trivially true. Suppose that it is true for $k-1$, i.e., each label has $O(n^{k-1})$ possible label positions for n labels with $k-1$ different heights. Then it suffices to show that the lemma is true for k.

Since labels are placed without overlaps, we consider the location of the bottom side of a label as a label position. Consider each label L_i, which is the i-th label from the bottom. Let η_k denote the k-th kind of label height. Suppose that there are m labels of height η_k below label L_i, and hence, there are $(i-1-m)$ labels of the other $k-1$ kinds of heights below label L_i. Since the modification of the ordering of the labels below label L_k does change the position of L_i, we push all these m labels of height η_k down to the bottom and move other labels upwards accordingly. That is, we consider the label stack below label L_i: from the bottom there are m labels of height η_k and then $(i-1-m)$ labels of the other $k-1$ kinds of heights.

Note that $m<i\le n$. For each possible $m=1,2,\cdots,i-1$, by inductive hypothesis, there are $O(n^{k-1})$ possible label positions for the $(i-1-m)$ labels of other kinds of heights. As a result, there are $O((i-1)\times n^{k-1})=O(n^k)$ possible label positions for each label L_i. □

Since the $O(n^4h)$ time algorithm in Theorem 5 considers h positions for the placement of labels, with Lemma 2, we see that the concerned problem can be solved in $O(n^4\cdot n^k)=O(n^{k+4})$ time. The result is stated as follows.

Theorem 6. *With respect to* 1.5*-side boundary labeling, both TLLM and TBM can be solved in* $O(n^{k+4})$ *time when the labels are of constant* k *different heights and ports are either fixed-ratio, fixed-position, or sliding.*

5 Conclusion

We have investigated the total leader length minimization problem and the bend number minimization problem for one-and-a-half-side boundary labeling under a variety of settings parameterized by the underlying label size (uniform vs. nonuniform) and port type (fixed-ratio, fixed-position, vs. sliding). It turns out the two problems under the uniform-label case are solvable in polynomial time, whereas the problems become NP-complete under the nonuniform-label case. A pseudo-polynomial time algorithm and a fixed-parameter algorithm have been proposed for those intractable problems. In addition, the case where indirected leaders are not beneficial was also identified.

References

1. Bekos, M., Kaufmann, M., Nöllenburg, M., Symvonis, A.: Boundary labeling with octilinear leaders. Algorithmica 57(3), 436–461 (2010)
2. Bekos, M., Kaufmann, M., Potina, K., Symvonis, A.: Area-feature boundary labeling. The Computer Journal 53(6), 827–841 (2009)
3. Bekos, M., Kaufmann, M., Symvonis, A., Wolff, A.: Boundary labeling: models and efficient algorithms for rectangular maps. Computational Geometry: Theory and Applications 36(3), 215–236 (2006)
4. Benkert, M., Haverkort, H., Kroll, M., Nöllenburg, M.: Algorithms for multi-criteria one-sided boundary labeling. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 243–254. Springer, Heidelberg (2008)
5. Formann, M., Wagner, F.: A packing problem with applications to lettering of maps. In: Proc. of the 7th Annual ACM Symposium on Computational Geometry (SoCG 1991), pp. 281–288. ACM Press, New York (1991)
6. Garey, M., Tarjan, R., Wilfong, G.: One-processor scheduling with symmetric earliness and tardiness penalties. Math. Oper. Res. 13, 330–348 (1988)
7. Lin, C.C.: Crossing-free many-to-one boundary labeling with hyperleaders. In: Proc. of 3rd IEEE Pacific Visualization Symposium (PacificVis 2010), pp. 185–192. IEEE Press, Los Alamitos (2010)
8. Lin, C.C., Kao, H.J., Yen, H.C.: Many-to-one boundary labeling. Journal of Graph Algorithms and Applications 12(3), 319–356 (2008)
9. Lin, C.C., Wu, H.Y., Yen, H.C.: Boundary labeling in text annotation. In: Proc. of 13th International Conference on Information Visualisation (IV 2009), pp. 110–115. IEEE CS Press, Los Alamitos (2009)
10. Wagner, F.: Approximate map labeling is in $\Omega(n \log n)$. Information Processing Letters 52(3), 161–165 (1994)
11. Wagner, F., Wolff, A.: Map labeling heuristics: Provably good and practically useful. In: Proc. of the 11th Annual ACM Symposium on Computational Geometry (SoCG 1995), pp. 109–118. ACM Press, New York (1995)

Approximation Algorithms for a Bi-level Knapsack Problem

Lin Chen and Guochuan Zhang

College of Computer Science, Zhejiang University, Hangzhou, 310027, China
zgc@zju.edu.cn

Abstract. In this paper, we consider a variant of knapsack problem. There are two knapsacks with probably different capacities, owned by two agents respectively. Given a set of items, each with a fixed size and a profit, the two agents select items and pack them into their own knapsacks under the capacity constraint. Same items can be packed simultaneously to different knapsacks. However, in this case the profit of such items can vary. One agent packs items into his knapsack to maximize the total profit, while another agent can only pack items into his knapsack as well but he cares the total profits of items packed into two knapsacks. The latter agent is a leader while the former is a follower. We aim at designing an approximation algorithm for the leader assuming that the follower is selfish. For different settings we provide approximation results.

Keywords: Bilevel knapsack problem, Approximation algorithms.

1 Introduction

Knapsack Problem (KP) is one of the classical NP-hard problems in combinatorial optimization and computer science. There exists a pseudo-polynomial time exact algorithm based on dynamic programming (DP) and a fully polynomial time approximation scheme (FPTAS) [4]. There has been a lot of generalization of the classical knapsack problem, one of which is the bi-level knapsack problem(BKP) introduced by Dempe and Richter [3]. In the bi-level knapsack problem, however, there are two decision makers, and they are in 2 different levels. For simplicity we can view the 2 decision makers as the leader and the follower. The leader controls the capacity of the follower's knapsack, and the follower then chooses items so that his own total profits are maximized (under the knapsack constraint). The goal, however, is to maximize the objective function of the leader, which is a linear combination of the total profits of items (items might have different values for the leader and the follower) and the capacity of the knapsack he gives his follower. It can be easily seen that here the maximum is not well-defined since there might be 2 solutions such that the follower's objective value is the same and yet the leader's objective value is different, and thus gives rise to two variants of this model [6], the optimistic case where once the follower encounters multiple optimum solutions, he chooses the one that maximizes the objective of the leader, and the pessimistic case where he chooses just the opposite. For both cases of this BKP model, Brotcorne et al. [1] proposed a dynamic programming algorithm. And for some comprehensive surveys on bi-level programming, we refer readers to [2].

In this paper, however, we deal with a different model. Again there are 2 decision makers (agents), a leader and a follower, and each of them has his own knapsack (might be of different capacity). Unlike the BKP models mentioned above, one can not control the other's knapsack, however, he can influence the profit of the items. As an example consider the following scenario. Regard each item as a project, and the two agents as investors. To put project i into investor j's portfolio it would cost him w_i, and reward him with a profit of $p_i + a_i$. Here $a_i = 0$ if he is the sole investor, and could be either positive or negative if project i is taken by both investors. It is clear the profit of a project depends on the decision of both investors. The follower is interested in his own profit while the leader aims at maximizing the total profits of two agents. In other words, the leader's objective is the social income. We aim to find a best solution for the leader. A simple interpretation of this model could be that, the leader is the government, and the follower is a company. The government wants to improve the social income by playing as a role in this game, although it owns a profit as a player in the game, however, its goal is the social income, and thus it plays as a 'selfless' player.

Our model is related to Wang et al. [8], whose focus is on the existence of a pure Nash equilibrium and the price of anarchy [5]. The generalization from 2 players to 2 groups is covered in [7]. Our model is different in the way that, the 2 investors in Wang et al. [8] are of the same level as they stands for 2 selfish players in a game, and yet in our model, one is a leader and the other is a follower.

In Section 2, we give some preliminaries and formulate our problem as a bi-level programming. We further define two different versions based on the item profits, namely a competitive version in which the profit of an item decreases if it is packed into both knapsacks, and a beneficial version otherwise. In Section 3, we consider the competitive version by giving a $2 + \epsilon$-approximation algorithm and a lower bound of 1.5. In Sections 4 and 5, we investigate the beneficial version with 2 different cases with respect to the capacities of the 2 knapsacks, and give a $(1 + \sqrt{2}) + \epsilon$-approximation algorithm and a 2-approximation algorithm, respectively. Concluding remarks are given in Section 6.

2 Preliminary and Problem Formulation

Let $U = \{1, 2, \cdots, n\}$ denote the set of items(projects) to be chosen from, with item i consuming a capital of w_i. Now there are two investors, just as we have mentioned, one is the leader (government) who wants to maximize the revenue of both (the total income), and the other is the follower (company) who only cares about its own profit. Note that item i will bring a revenue of p_i if it is chosen by either one, and if it is chosen by both, then each one gains $p_i + a_i$ (a_i could be positive or negative).

Let $x_i, y_i \in \{0,1\}, i = 1, 2, \cdots, n$ denote the choices of the leader and the follower for item i respectively, and W_1 and W_2 denote the total capitals of them respectively. This Stackelberg problem can be modeled as a bi-level programming as follows.

$$\begin{array}{lll} \underset{x,y\in M(x)}{max} & f^1(x,y) = \sum_{i=1}^{n} p_i(x_i + y_i) + 2\sum_{i=1}^{n} a_i x_i y_i & \\ s.t. & \sum_{i=1}^{n} w_i x_i \le W_1, \qquad x_i \in \{0,1\} & \end{array}$$

where $M(x)$ is the set of optimum solutions of the following $0-1$ integer programming for any fixed x.

$$\begin{array}{ll} max & f^2(x,y) = \sum_{i=1}^{n} p_i y_i + \sum_{i=1}^{n} a_i x_i y_i \\ s.t. & \sum_{i=1}^{n} w_i y_i \le W_2, \qquad y_i \in \{0,1\} \end{array}$$

Remarks.

1. In our problem, we assume that the leader makes its choice first, and then the follower will solve the corresponding knapsack problem with respect to its objective function $f^2(x,y)$, and we further assume that the follower can always find an optimum solution for this knapsack problem.
2. There is one problem with the above programming. It is actually not well-defined in the sense that for some x, $M(x)$ might contain more than one elements, and the objective value of the leader thus depends on the choice of the follower. A weak Stackelberg problem assumes that when facing multiple choices, the follower will always return a solution that minimizes the leader's objective value while a strong Stackelberg problem assumes just the opposite. However, in reality the reaction pattern of a follower is often his private information and is unknown to the leader. Thus in our problem, we assume that when multiple choices are available to the follower, he will choose one arbitrarily.

   Since we focus on approximation algorithms whose analysis is based on worst case performance, we assume that when analyzing any given algorithm, the follower always returns a solution that minimizes the leader's objective value. Yet the optimum solution (x_i^*, y_i^*) is the one that achieves the largest possible value of $f^1(x,y)$, which is the optimum objective value of the strong Stackelberg problem, and we call it the cooperative optimum solution.

In the following sections, we will focus on two special cases, the competitive version (i.e. $a_i \le 0, \forall i$) and the beneficial version (i.e. $a_i \ge 0, \forall i$), and for the beneficial version we give two different algorithms under conditions $W_1 \leq W_2$ and $W_1 > W_2$, respectively.

For simplicity, in the following part, we may replace $\sum_{i=1}^{n}$ by $\sum$, and omit the constraint $x_i, y_i \in \{0,1\}$.

3 The Competitive Version

Clearly $a_i \leq 0$ $(\forall i)$ implies that items chosen by the leader become less attractive to both sides. For simplicity, we replace a_i by $-a_i$.Then we have

$$\begin{aligned}
&\max_{x,y\in M(x)} \quad f^1(x,y) = \sum_{i=1}^{n} p_i(x_i+y_i) - 2\sum_{i=1}^{n} a_i x_i y_i \\
&s.t. \quad \sum_{i=1}^{n} w_i x_i \leq W_1, \qquad x_i \in \{0,1\} \\
&M(x) \;:\; \max \quad f^2(x,y) = \sum_{i=1}^{n} p_i y_i - \sum_{i=1}^{n} a_i x_i y_i \\
&\qquad s.t. \quad \sum_{i=1}^{n} w_i y_i \leq W_2, \qquad y_i \in \{0,1\} \\
&\qquad (p_i, w_i, W_1, W_2 > 0, 0 \leq a_i \leq p_i, \forall i)
\end{aligned}$$

In this section we will give a pseudo-polynomial time 2-approximation algorithm as well as a polynomial time $(2+\varepsilon)$-approximation algorithm for our problem, and also show that there is a lower bound of 1.5.

3.1 Algorithm Alg_c

Algorithm Alg_c is as follows.

Step 1. Solve knapsack problem (1)

$$\begin{aligned}
\max \quad & \sum p_i y_i \\
s.t. \quad & \sum w_i y_i \leq W_2
\end{aligned} \tag{1}$$

Let y_{i1} be an optimum or an approximation solution.

Step 2. Solve knapsack problem (2)

$$\begin{aligned}
\max \quad & \sum p_i x_i - 2\sum a_i x_i y_{i1} \\
s.t. \quad & \sum w_i x_i \leq W_1
\end{aligned} \tag{2}$$

Let x_{i1} be an optimum or an approximation solution.

Step 3.Output x_{i1}.

We will show that, if we use a pseudo-polynomial algorithm to solve knapsack problems (1) and (2) exactly, then the above algorithm is a pseudo-polynomial time 2-approximation algorithm, and if we use instead an FPTAS to achieve a $(1+\varepsilon)$-approximation solution for (1) and (2), then the above algorithm is a polynomial time $2(1+\varepsilon)/(1-\varepsilon)$-approximation algorithm. For simplicity we only consider the use of FPTAS in the following subsection, and the analysis for the use of a pseudo-polynomial time algorithm is just similar.

3.2 Analysis of the Algorithm

Let x_{i1} be the choice of the leader, and y'_{i1} be any corresponding choice of the follower, we will prove that $OPT = \sum p_i(x_i^* + y_i^*) - 2\sum a_i x_i^* y_i^*$ is bounded by $2(1+\varepsilon)/(1-\varepsilon)$ times of $\sum p_i(x_{i1} + y'_{i1}) - 2\sum a_i x_{i1} y'_{i1}$

Lemma 1. *If x_{i1}, y_{i1} are $(1+\varepsilon)$- approximation solutions of (1) and (2), then*

$$OPT \le 2(1+\varepsilon)(\sum p_i x_{i1} + \sum p_i y_{i1} - 2\sum a_i x_{i1} y_{i1})$$

Proof. Obviously we have $(1+\varepsilon)\sum p_i y_{i1} \ge \sum p_i y_i^*$, $\sum p_i x_{i1} - 2\sum a_i x_{i1} y_{i1} \ge 0$. Thus $(1+\varepsilon)(\sum p_i y_{i1} + \sum p_i x_{i1} - 2\sum a_i x_{i1} y_{i1}) \ge \sum p_i y_i^*$.

Let I be the set of subscripts of those y_{i1} that are equal to 1, i.e. $I = \{i | y_{i1} = 1\}$, then $(1+\varepsilon)(\sum p_i x_{i1} - 2\sum a_i x_{i1} y_{i1}) \ge \underset{\sum w_i x_i \le W_1}{max} \sum_{i \notin I} p_i x_i$.

$$\begin{aligned}(1+\varepsilon)(\sum p_i x_{i1} + \sum p_i y_{i1} - 2\sum a_i x_{i1} y_{i1}) &\ge \sum_{i\in I} p_i + \underset{\sum w_i x_i \le W_1}{max} \sum_{i\notin I} p_i x_i \\ &\ge \underset{\sum w_i x_i \le W_1}{max} \sum p_i x_i \\ &\ge \sum p_i x_i^*\end{aligned}$$

thus

$$2(1+\varepsilon)(\sum p_i x_{i1} + \sum p_i y_{i1} - 2\sum a_i x_{i1} y_{i1}) \ge \sum p_i y_i^* + \sum p_i x_i^* \ge OPT.$$

Theorem 1. *Let y'_{i1} be any corresponding choice of the follower when the leader chooses according to x_{i1}, then*

$$OPT \le 2(1+\varepsilon)/(1-\varepsilon)(\sum p_i(x_{i1} + y'_{i1}) - 2\sum a_i x_{i1} y'_{i1}).$$

The proof is omitted.

3.3 A Lower Bound

Consider the following program:

$$\begin{array}{lll} \underset{x,y\in M(x)}{max} & f^1(x,y)=(x_1+y_1)+2(x_2+y_2)-2x_2y_2 & \\ s.t. & 2x_1+x_2\le 1 & \\ M(x)\ : & max \quad f^2(x,y)=y_1+2y_2-x_2y_2 & \\ & \quad s.t. \quad 2y_1+y_2\le 2 & \end{array}$$

Any algorithm for this problem will return one of the following 2 solutions (in the collum of (x_1, x_2)), and suppose the follower chooses the corresponding solution in the collum for (y_1, y_2) (it's easy to verify that they are optimum solutions for the corresponding knapsack problems).

(x_1, x_2)	(y_1, y_2)	$x_1 + y_1 + 2(x_2 + y_2) - 2x_2y_2$
(0,1)	(0,1)	2
(0,0)	(0,1)	2

The cooperative optimum solution of this problem, however, is $(x_1, x_2) = (0, 1)$, $(y_1, y_2) = (1, 0)$, then $x_1 + y_1 + 2(x_2 + y_2) - 2x_2y_2 = 3$. Thus a lower bound of this problem is 1.5.

4 The Beneficial Version with $W_1 > W_2$

Bilevel integer programming for this problem is as follows.

$$\begin{array}{lll} \underset{x,y\in M(x)}{max} & f^1(x,y)=\sum_{i=1}^n p_i(x_i+y_i)+2\sum_{i=1}^n a_ix_iy_i & \\ s.t. & \sum_{i=1}^n w_ix_i\le W_1, \qquad x_i\in\{0,1\} & \\ M(x)\ : & max \quad f^2(x,y)=\sum_{i=1}^n p_iy_i+\sum_{i=1}^n a_ix_iy_i & \\ & \quad s.t. \quad \sum_{i=1}^n w_iy_i\le W_2, \qquad y_i\in\{0,1\} & \\ & (p_i, w_i, W_1, W_2, a_i\ge 0, p_i+a_i>0, \forall i, W_1>W_2) & \end{array}$$

In this section we will give a pseudo-polynomial time $(1+\sqrt{2})$-approximation algorithm as well as a polynomial time $(1+\sqrt{2}+\varepsilon)$-approximation algorithm for this problem, and also show that there is a lower bound of 1.5.

4.1 Algorithm Alg_{m1}

Algorithm Alg_{m1} is as follows.

Step 1. Solve knapsack problem (3)

$$\begin{aligned} max & \sum p_i x_i \\ s.t. & \sum w_i x_i \le W_1 \end{aligned} \tag{3}$$

Let x_{i1} be an optimum or an approximation solution. then solve knapsack problem (4)

$$\begin{aligned} max & \sum p_i y_i + \sum a_i x_{i1} y_i \\ s.t. & \sum w_i y_i \le W_2 \end{aligned} \tag{4}$$

Let y_{i1} be an optimum or an approximation solution.

Step 2. Solve knapsack problem (5)

$$\begin{aligned} max & \sum (p_i + a_i) x_i \\ s.t. & \sum w_i x_i \le W_2 \end{aligned} \tag{5}$$

Let x_{i2} be an optimum or an approximation solution, then solve knapsack problem (6)

$$\begin{aligned} max & \sum p_i y_i + \sum a_i x_{i2} y_i \\ s.t. & \sum w_i y_i \le W_2 \end{aligned} \tag{6}$$

Let y_{i2} be an optimum or an approximation solution.

Step 3. If $\sqrt{2}(\sum p_i x_{i1} + \sum p_i y_{i1} + \sum a_i x_{i1} y_{i1}) \ge \sum p_i y_{i2} + 2\sum a_i x_{i2} y_{i2}$ output x_{i1}, otherwise output x_{i2}.

Again if we use a pseudo-polynomial time algorithm to solve knapsack problem (3), (4), (5) and (6) exactly, then the above algorithm is a pseudo-polynomial time $(1+\sqrt{2})$-approximation algorithm, and if we use an FPTAS to achieve a $(1+\varepsilon)$-approximation solution for those knapsack problems, then the above algorithm is a polynomial time $(1+\sqrt{2})(1+\varepsilon)^3$-approximation algorithm. In the subsection follows again we only focus on the use of the FPTAS.

4.2 Analysis of the Algorithm

When the leader chooses x_{i1} or x_{i2}, let y'_{i1} or y'_{i2} be any corresponding choice of the follower, we'll prove that $OPT = \sum p_i(x_i^* + y_i^*) + 2\sum a_i x_i^* y_i^*$ is bounded by $(1+\sqrt{2})(1+\varepsilon)^3$ times of either $\sum p_i(x_{i1} + y'_{i1}) + 2\sum a_i x_{i1} y'_{i1}$ or $\sum p_i(x_{i2} + y'_{i2}) + 2\sum a_i x_{i2} y'_{i2}$.

Lemma 2. *If $x_{i1}, y_{i1}, x_{i2}, y_{i2}$ are $(1+\varepsilon)$-approximation solutions for knapsack problems (3), (4), (5) and (6), then*

$$OPT \le (1+\varepsilon)^3(\sum p_i x_{i1} + \sum p_i y_{i1} + \sum a_i x_{i1} y_{i1} + \sum p_i y_{i2} + 2\sum a_i x_{i2} y_{i2})$$

Proof. Note that

$$\begin{aligned}
&(a). \quad (1+\varepsilon)\sum p_i x_{i1} \geq \sum p_i x_i^* \\
&(b). \ (1+\varepsilon)^2(\sum p_i y_{i2} + \sum a_i x_{i2} y_{i2}) \geq (1+\varepsilon)(\sum p_i x_{i2} + \sum a_i x_{i2}) \\
&\qquad \geq \sum p_i y_i^* + \sum a_i y_i^* \\
&\qquad \geq \sum p_i y_i^* + \sum a_i x_i^* y_i^* \\
&(c). \quad (1+\varepsilon)^2 \sum a_i x_{i2} y_{i2} \geq \sum a_i x_i^* y_i^* - (1+\varepsilon)^2 \sum p_i y_{i2} \\
&(d). \ (1+\varepsilon)(\sum p_i y_{i1} + \sum a_i x_{i1} y_{i1}) \geq \sum p_i y_{i2} + \sum a_i x_{i1} y_{i2} \\
&\qquad \geq \sum p_i y_{i2}
\end{aligned}$$

Let $(a)+(b)+(c)+(d)*(1+\varepsilon)^2$, and we get what we expect.

Theorem 2. *If $x_{i1}, y_{i1}, x_{i2}, y_{i2}$ are $1+\varepsilon$-approximation solutions for knapsack problems (3), (4), (5) and (6), and let y'_{i1} or y'_{i2} be any choice of the follower when the leader chooses x_{i1} or x_{i2}, $\alpha > 0$, we have*

Claim 1. If $\alpha(\sum p_i x_{i1} + \sum p_i y_{i1} + \sum a_i x_{i1} y_{i1}) \geq \sum p_i y_{i2} + 2\sum a_i x_{i2} y_{i2}$ then

$$OPT \leq (1+\alpha)(1+\varepsilon)^3(\sum p_i x_{i1} + \sum p_i y'_{i1} + 2\sum a_i x_{i1} y'_{i1}).$$

Claim 2. If $\alpha(\sum p_i x_{i1} + \sum p_i y_{i1} + \sum a_i x_{i1} y_{i1}) \leq \sum p_i y_{i2} + 2\sum a_i x_{i2} y_{i2}$ then

$$OPT \leq (1+\frac{2}{\alpha})(1+\varepsilon)^3(\sum p_i x_{i2} + \sum p_i y'_{i2} + 2\sum a_i x_{i2} y'_{i2}).$$

The proof is omitted.

4.3 A Lower Bound

Consider the following case:

$$\begin{aligned}
\max_{x,y\in M(x)} \quad & f^1(x,y) = (x_1+y_1) + (x_3+y_3) + 2x_2y_2 \\
s.t. \quad & 2x_1 + 3x_2 + 4x_3 \leq 5 \\
M(x) \ : \ \max \quad & f^2(x,y) = y_1 + y_3 + x_2y_2 \\
s.t. \quad & 2y_1 + 3y_2 + 4y_3 \leq 4
\end{aligned}$$

Any algorithm for this problem will return one of the following 5 solutions (in the collum of (x_1, x_2, x_3)), suppose the follower chooses the corresponding solution in the collum for (y_1, y_2, y_3) (it's easy to verify that they are optimum solutions for the corresponding knapsack problems).

(x_1, x_2, x_3)	(y_1, y_2, y_3)	$x_1 + y_1 + x_3 + y_3 + 2x_2y_2$
(0,0,0)	(0,0,1)	1
(1,0,0)	(0,0,1)	2
(0,1,0)	(0,0,1)	1
(0,0,1)	(0,0,1)	2
(1,1,0)	(0,0,1)	2

And the cooperative optimum solution is $(x_1, x_2, x_3) = (1, 1, 0), (y_1, y_2, y_3) = (0, 1, 0)$ with the object value 3, thus one lower bound for this problem is 1.5.

5 The Beneficial Version with $W_1 \leq W_2$

The bi-level integer programming for this problem is as follows.

$$\begin{aligned}
&max \quad f^1(x,y) = \sum p_i(x_i + y_i) + 2\sum a_i x_i y_i \\
&s.t. \quad \sum w_i x_i \leq W_1 \\
&M(x) : max \quad f^2(x,y) = \sum p_i y_i + \sum_{i=1}^{n} a_i x_i y_i \\
&\qquad s.t. \quad \sum w_i y_i \leq W_2 \\
&\qquad (p_i, w_i, W_1, W_2, a_i \geq 0, p_i + a_i > 0, \forall i, W_1 \leq W_2)
\end{aligned}$$

In this section we will give a pseudo-polynomial time 2-approximation algorithm as well as a polynomial time $2 + \varepsilon$-approximation algorithm for this problem, and also show that there is a lower bound of 2.

Before the main algorithm Alg_{m2} is given for this problem, we first give algorithms for the following subproblem which will be used in the main algorithm.

5.1 A Subproblem and the Corresponding Algorithm

Integer programming for this subproblem is as follows:

$$\begin{aligned}
max \quad & \sum_{i=1}^{n} p_i y_i + \sum a_i x_i y_i \\
s.t. \quad & \sum_{i=1}^{n} w_i x_i \leq W_1 \\
& \sum_{i=1}^{n} w_i y_i \leq W_2
\end{aligned}$$

Again, constraints $x_i, y_i \in \{0, 1\}, i = 1, 2, \cdots n$ is omitted.

We will give a pseudo-polynomial time exact algorithm Alg_{sub} as well as a polynomial time approximation scheme $Alg_{sub}^{'}(\varepsilon)$ which will return a solution x_i, y_i that satisfy:

$$OPT^{'} \leq (1 + \varepsilon)(\sum p_i x_i + \sum p_i y_i + \sum a_i x_i y_i),$$

where $OPT^{'}$ is the optimum object value of this subproblem.

Note that $Alg_{sub}^{'}(\varepsilon)$ is not a PTAS for this subproblem, but this result is enough as we set out to design an $(2 + \varepsilon)$-approximation algorithm for the main problem.

As the algorithm $Alg_{sub}^{'}(\varepsilon)$ involves a lot of enumerations, we omit the proof.
Dynamic programming Alg_{sub}:
Define $Z_k(a,b)$ recursively as follows.

$$Z_{k+1}(a,b) = max\{Z_k(a,b), Z_k(a-w_{k+1}, b-w_{k+1})+p_{k+1}+a_{k+1}, Z_k(a, b-w_{k+1})+p_{k+1}\}$$

$(a,b,k \in \mathbf{Z}, a \le W_1, b \le W_2, 1 \le k \le n-1)$
The initial condition is

$$Z_1(a,b) = \begin{cases} 0, & b < w_1 \\ p_1, & b \ge w_1, a < w_1 \\ p_1 + a_1, & b \ge w_1, a \ge w_1 \end{cases}$$

$$\forall k, \quad Z_k(a,b) = -\infty, \quad \text{if} \quad a < 0 \quad \text{or} \quad b < 0$$

The algorithm stops as we compute $Z_n(W_1, W_2)$, and it's easy to verify that the computational complexity is $O(nW_1W_2)$.
The analysis of this algorithm is omitted.

5.2 Algorithm Alg_{m2}

We describe Algorithm Alg_{m2} as follows.
Step 1. Solve knapsack problem (7)

$$\begin{aligned} max \quad & \sum (p_i + a_i)x_i \\ s.t. \quad & \sum w_i x_i \le W_1 \end{aligned} \tag{7}$$

Let x_{i1} be an optimum or an approximation solution.
Step 2. Solve the following subproblem (8) using Alg_{sub} or $Alg_{sub}^{'}(\varepsilon)$

$$\begin{aligned} max \quad & \sum p_i y_i + \sum a_i x_i y_i \\ s.t. \quad & \sum w_i x_i \le W_1 \\ & \sum w_i y_i \le W_2 \end{aligned} \tag{8}$$

Let x_{i2}, y_{i2} be an optimum or an approximation solution.
Step 3. If $\sum (p_i + a_i)x_{i1} \ge \sum p_i x_{i2} + \sum p_i y_{i2} + \sum a_i x_{i2} y_{i2}$, output$x_{i1}$, otherwise output x_{i2}.

Similarly, If we use a pseudo-polynomial algorithm to solve knapsack problem (7) exactly, and Alg_{sub} to solve problem (8), then the above algorithm is a pseudo-polynomial time 2-approximation algorithm, and if we use an FPTAS to achieve a $(1+\varepsilon)$-approximation solution for knapsack problem (7) and $Alg_{sub}^{'}(\varepsilon)$ for problem (8), then the above algorithm is a polynomial time $(2+2\varepsilon)$-approximation algorithm. In the subsection follows we again only focus on the polynomial time algorithm.

5.3 Analysis of the Algorithm

When the leader chooses x_{i1} or x_{i2}, let y'_{i1} or y'_{i2} be any choice of the follower, we'll prove that $OPT = \sum p_i(x_i^* + y_i^*) + 2\sum a_i x_i^* y_i^*$ is bounded by $2(1+\varepsilon)$ times of either $\sum p_i(x_{i1} + y'_{i1}) + 2\sum a_i x_{i1} y'_{i1}$ or $\sum p_i(x_{i2} + y'_{i2}) + 2\sum a_i x_{i2} y'_{i2}$.

Lemma 3. *If $x_{i1}, y_{i1}, x_{i2}, y_{i2}$ are $(1+\varepsilon)$-approximation solutions for problems (7) and (8), then*

$$OPT \le (1+\varepsilon)(\sum p_i x_{i1} + \sum a_i x_{i1} + \sum p_i x_{i2} + \sum p_i y_{i2} + \sum a_i x_{i2} y_{i2}).$$

Proof. Note that

$$\begin{aligned}(1+\varepsilon)(\sum p_i x_{i1} + \sum a_i x_{i1}) &\ge \sum p_i x_i^* + \sum a_i x_i^* \\ &\ge \sum p_i x_i^* + \sum a_i x_i^* y_i^*\end{aligned}$$

and

$$(1+\varepsilon)(\sum p_i x_{i2} + \sum p_i y_{i2} + \sum a_i x_{i2} y_{i2}) \ge \sum p_i y_i^* + \sum a_i x_i^* y_i^*.$$

Summing them up we arrive at what we want.

Theorem 3. *If $x_{i1}, y_{i1}, x_{i2}, y_{i2}$ are $(1+\varepsilon)$-approximation solutions for problems (7) and (8), and let y'_{i1} or y'_{i2} be any corresponding choice of the follower respectively, then*

Claim 1. If $\sum(p_i + a_i)x_{i1} \ge \sum p_i x_{i2} + \sum p_i y_{i2} + \sum a_i x_{i2} y_{i2}$, then $OPT \le 2(1+\varepsilon)\sum(p_i + a_i)x_{i1}$.

Claim 2. If $\sum(p_i + a_i)x_{i1} < \sum p_i x_{i2} + \sum p_i y_{i2} + \sum a_i x_{i2} y_{i2}$, then $OPT \le 2(1+\varepsilon)(\sum p_i x_{i2} + \sum p_i y'_{i2} + \sum a_i x_{i2} y'_{i2})$

We omit the proof.

5.4 A Lower Bound

Consider the following case:

$$\begin{array}{lll} \max\limits_{x,y\in M(x)} & f^1(x,y) = x_2 + y_2 + 2x_1y_1 & \\ s.t. & x_1 + 2x_2 \le 1 & \\ M(x) \;:\; & \max & f^2(x,y) = y_2 + x_1y_1 \\ & s.t. & y_1 + 2y_2 \le 2 \end{array}$$

Any algorithm for this problem will return one of the following 2 solutions (in the collum of (x_1, x_2)), suppose the follower chooses the corresponding solution in the collum for (y_1, y_2) (it's easy to verify that they are optimum solutions for the corresponding knapsack problems).

(x_1, x_2)	(y_1, y_2)	$x_2 + y_2 + 2x_1y_1$
(1,0)	(0,1)	1
(0,0)	(0,1)	1

And the cooperative optimum solution is $(x_1, x_2) = (1, 0), (y_1, y_2) = (1, 0)$ with the object value 2, thus one lower bound for this problem is 2.

6 Conclusions

In this paper, we consider the portfolio problem in the real world that involves two decision makers with one focuses on the total profits. We formulate this problem as a bi-level knapsack problem, and provide approximation algorithms under different settings. The corresponding lower bounds are also analyzed. It is interesting to extend our work to incorporate a general situation with more players.

References

1. Brotcorne, L., Hanafi, S., Mansi, R.: A dynamic programming algorithm for the bi-level knapsack problem. Operations Research Letters 37, 215–218 (2009)
2. Colson, B., Marcotte, P., Savard, G.: Bilevel programming, A survey. 4OR: A Quarterly Journal of Operations Research 3(2), 87–107 (2005)
3. Dempe, S., Richter, K.: Bilevel programming with Knapsack constraint. Central European Journal of Operations Research 8, 93–107 (2000)
4. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM 22, 363–468 (1975)
5. Koutsoupias, E., Papadimitriou, C.H.: Worst-case equilibria. In: Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science, pp. 404–413 (1999)
6. Loridan, P., Morgan, J.: Weak via strong Stackelberg problem: Newresults. Journal of Global Optimization 8, 263–287 (1996)
7. Wang, Z., Xing, W., Fang, S.C.: Two-group knapsack game. Theoretical Computer Science 411, 1094–1103 (2010)
8. Wang, Z., Xing, W.: Two-person knapsack game. Journal of Industrial and Management Optimization 6(4), 847–860 (2010)

On the Surface Area of the Asymmetric Twisted Cube

Eddie Cheng[1], Qiu Ke[2], and Zhizhang Shen[3]

[1] Mathematics and Statistics, Oakland University, USA
echeng@oakland.edu

[2] Dept of Computer Science, Brock University, Canada
kqiu@brocku.ca

[3] Dept. of Computer Science and Technology, Plymouth State University, USA
zshen@plymouth.edu

Abstract. We derive a surface area result for the asymmetric twisted cube, provide closed-form expressions for such results in terms of some exemplary centers, and start to make an accurate analysis of its associated average distance measurement.

Keywords: Interconnection networks, broadcasting, twisted cube, surface area, average distance.

1 Introduction

Given a vertex u in a graph G, a question one may ask is *how many vertices are at distance* i *from* u for $i \in [0, D(G)]$, where $D(G)$ stands for the diameter of G. This quantity is referred to, in the literature, as the "Whitney numbers of the second kind of the poset" [13], the "surface area of a vertex with radius i" [12], and "distance distribution of nodes" [17]. In this paper, we refer to this quantity as the *surface area with radius* i, *centered at* u, denoted as $B_{G,u}(i)$.

The surface area of a network can find several applications in evaluating network performance. In particular, it can be used to derive the *average distance* of a network structure. The *average distance from* u *to all the vertices in* G is often defined as $\frac{\sum_{v \in V(G)} d_G(u,v)}{|V(G)|}$, namely, $\frac{\sum_{i=0}^{D(G)} i * B_{G,u}(i)}{|V(G)|}$, measuring the broadcasting behavior of G in terms of u, while the surface area result itself can be used to characterize the k-neighborhood broadcasting behavior of G from u [8]. Other applications of surface area results include identification of spanning trees and resource placement in network structures. As a result, this surface area problem has been studied for a variety of network structures, including the rotator graph, the star graph, the k-ary n-cube, the (n, k)-star graph, and the arrangement graph. (For the solutions to this problem for the aforementioned and other graphs, readers are referred to [12,15] and the references cited within.)

It is generally difficult to derive surface area results for non vertex-symmetric structures, where the surface area varies from center to center. For one such example, readers are referred to [4] for general surface area results for the symmetric torus structure and the asymmetric mesh structure.

Several hypercube variants have been suggested, including augmented cubes, cross cubes, Möbius cubes, and twisted cubes. The chief attraction of the twisted cubes is that, while having the same number of edges, its diameter is only half of that of the hypercube of the same dimension, thus cutting down the routing cost. Abraham *et al* considered this "strong point" in [1] by discussing the static asymmetry in the twisted cube topology and its impact on inter-vertex distances, i.e., surface area, and average distance. Various topological properties of the twisted cube structure, including connectivity, wide diameter, fault diameter and pancyclicity properties, are studied in [3], and more recent research results have appeared in [11,18,7,2]. But, to the best of our knowledge, no analysis has ever been made to obtain a general result on the surface area of this vertex asymmetric class of twisted cubes. Lack of such a result makes an accurate analysis of the average distance measurement difficult, if not impossible, for the twisted cubes.

In this paper, we derive the surface area result for the twisted cube. Besides solving a specific surface area problem, we believe this result also sheds new light on solving similar problems in terms of other asymmetric structures. To make this paper self contained, we provide a brief overview of the necessary terms and basic results related to the twisted cube in the next section, derive a general surface result in Section 3, and provide closed-form expressions relative to some specific centers in Section 4. We then present surface areas, as well as the average distances, of twisted cubes of varying sizes, relative to some of the exemplary centers, in Section 5, and finally end this paper with some concluding remarks in Section 6.

2 Twisted Cube and Its Routing

The vertex set of the *twisted-cube* TQ_d, as coined in [10], is the set of all the binary strings of length $d, d \geq 1$, d odd, although the case of d being even has also been defined [10,7]. Let $d = 2m + 1, m \geq 0$.

Let $u \in TQ_d$, $u = u_{2m}u_{2m-1} \cdots u_{2j}u_{2j-1} \cdots u_0$, for all $j \in [1, m]$, $P_j(u)$, the *parity function* associated with the *double bit* $u_{2j}u_{2j-1}$, is defined as $u_{2j-2} \oplus \cdots \oplus u_0$, where '$\oplus$' stands for the exclusive-or operation.

Besides $u_{2m}u_{2m-1} \cdots \overline{u_0}$, u is always connected to $u_{2m}u_{2m-1} \cdots \overline{u_{2j}}u_{2j-1} \cdots u_0$, for $j \in [1, m]$. Moreover, u is also connected to either $u_{2m}u_{2m-1} \cdots \overline{u_{2j}u_{2j-1}} \cdots u_0$, when $P_j(u) = 0$; or $u_{2m}u_{2m-1} \cdots u_{2j}\overline{u_{2j-1}} \cdots u_0$, when $P_j(u) = 1$. It is thus clear that TQ_d contains $d2^{d-1}$ edges, the same as its hypercube counterpart of the same dimension. It is also known that the diameter of TQ_d is $\frac{d+1}{2}$ [10]. Figure 1 shows TQ_1 and TQ_3.

For example, in TQ_3, $u = 01\,0$, besides being connected to $01\,1$ and $11\,0$, is also connected to $10\,0$, since $P_1(u) = 0$; while $v = 01\,1$, besides $01\,0$ and $11\,1$, is connected to $00\,1$, since $P_1(v) = 1$.

The following routing algorithm is given in [3,7], equivalent to the original one as given in [10], for routing a message from a vertex c to another vertex u in TQ_d, at an intermediate vertex z, where for $j \in [1, m]$, u_{2j} is referred to as the

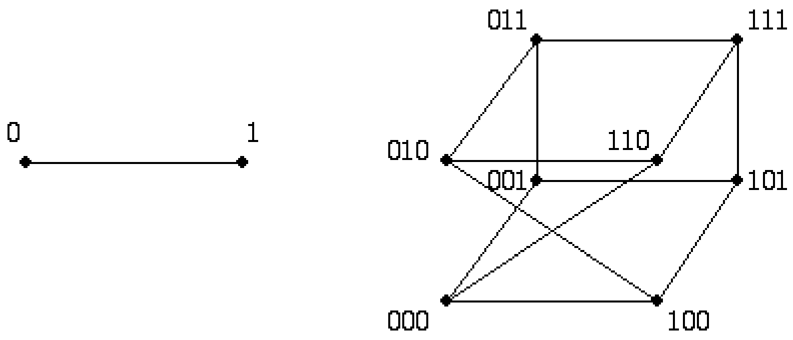


Fig. 1. Twisted cube of small dimensions

left part of the j^{th} *double bit of* u, namely, $u_{2j}u_{2j-1}$. Note that u_0, containing only one bit, is also referred to as a double bit for convenience.

R_1. If $z = u$, we are done.
R_2. Otherwise, repeatedly correct the leftmost differing j^{th} double bit between z and u that may be corrected by a single link traversal from vertex z and route the message via this link.
R_3. If the previous steps fail, find the rightmost differing j^{th} double bit between z and u and route the message to link $2j$ which will correct the left part of that double bit (but not completely correct the double bit itself.)

For example, when routing a message from $u = 01\,11\,00\,11\,1$ to $v = 00\,00\,00\,00\,1$, since $P_4(u) = 1$, we can correct the right part of u_4 to reach $w = 00\,11\,00\,11\,1$. We then get stuck since neither w_3 nor w_1 can be corrected to their corresponding double bits in v, because the respective parity function "blocks" such corrections. We thus have to apply R_3 to w_1, the rightmost differing double bit, when we only change its left part, to reach $x = 00\,11\,00\,01\,1$, which changes the values of all the parity functions associated with the preceding double bits, particularly those blocking parity functions, in this case, $P_3(u)$, the parity function associated with x_3. We can thus correct x_3 to 00 to reach $y = 00\,00\,00\,01\,1$, and finally correct y_1, the partially corrected rightmost double bit, to reach $v = 00\,00\,00\,00\,1$. In other words,

$$u = 01\,11\,00\,11\,1 \rightarrow 00\,11\,00\,11\,1 \overset{R_3}{\rightarrow} 00\,11\,00\,01\,1 \rightarrow 00\,00\,00\,01\,1 \rightarrow 00\,00\,00\,00\,1 = v.$$

It is proved in [10] that the above routing algorithm is correct and always leads to a minimum routing path between c and u. It is clear that, when no R_3 rule is applied, the number of differing double bits equals the number of corrections, thus the distance by the minimality of the routing algorithm. On the other hand, if, at some point, R_2 cannot be applied to any of the differing double bits, thus R_3 has to be applied to the left part of the rightmost differing double bit. Once this is done, the parity functions associated with all the remaining differing double bits will be changed to their opposite, respectively, thus all of them can now be corrected with R_2. Hence, R_3 needs to be applied exactly once in this latter case, when it takes one more move to correct the partially corrected rightmost double bit to its corresponding bit in c to complete the process, a total of two moves for the rightmost differing double bit, and one move for the other differing double bits. It is then clear that, for this latter case, the number of moves, i.e., the distance between the two vertices, equals the number of differing double bits plus one. Thus, we have the following result.

Proposition 1. *Let $h_d(u, v)$ stand for the number of the different double bits between two vertices $u, v \in TQ_d$, then*

$$d_{TQ_d}(u, v) = \begin{cases} h_d(u, v) & \text{if } R_3 \text{ is not applied;} \\ h_d(u, v) + 1 & \text{otherwise.} \end{cases}$$

For example, in the above example of routing from $u = 01\,11\,00\,11\,1$ to $v = 00\,00\,00\,00\,1$, although $h_d(u, v) = 3$, since R_3 is needed, their distance is 4.

We note that the above proposition implies the following "coarser" result [3, Theorem 2]:

Proposition 2. *Let $h_d(u, v)$ stand for the number of the different double bits between two vertices $u, v \in TQ_d$, then*

$$h_d(u, v) \leq d_{TQ_d}(u, v) \leq h_d(u, v) + 1.$$

It turns out that Proposition 1 opens the door for the derivation of the general surface area result for the twisted cube, which we will explore in the next section.

3 A General Surface Area Result for the Twisted Cube

Let $B_{TQ_d,c}(i)$ refer to the surface area of TQ_d, $d = 2m+1, m \geq 0$, with radius $i \in \left[0, \frac{d+1}{2}\right]$, centered at $c \in TQ_d$. It is clear that $B_{TQ_d,c}(0) = 1$, and $B_{TQ_d,c}(1) = d$, for any *center vertex* c. In this section, we derive $B_{TQ_d,c}(i), i \in [2, m+1]$, centered at any vertex c in TQ_d.

Let $u = u_{2m}u_{2m-1} \cdots u_{2j}u_{2j-1} \cdots u_0$, for convenience, we also write $u = u_m \cdots u_j \cdots u_1u_0$, where $u_j, j \in [1, m]$, refers to the j^{th} double bit of u, and similarly denote $P_j(u), j \in [1, m]$, as the parity function associated with u_j.

Let $c \in TQ_d, c = c_m \cdots c_j \cdots c_1c_0 \in TQ_d$, be a center vertex in TQ_d, when we say in the following that u_j *cannot be corrected to c_j because of its associated parity function,* we mean that, *when $P_j(u) = 0$, $u_j = c_{2j}\overline{c_{2j-1}}$, and when $P_j(u) = 1$, $u_j = \overline{c_{2j}c_{2j-1}}$,* and we will call such a parity function a *blocking* one. When we say that u_j *can be corrected to c_j because of its associated parity function,* we mean that, *when $P_j(u) = 0$, either $u_j = \overline{c_{2j}}c_{2j-1}$, or $u_j = \overline{c_{2j}c_{2j-1}}$; and when $P_j(u) = 1$, either $u_j = \overline{c_{2j}}c_{2j-1}$, or $u_j = c_{2j}\overline{c_{2j-1}}$.*

Incidentally, we note that when u_j can not be corrected because of its associated parity function, if the left part of u_j is changed to its opposite, this half corrected double bit can then be corrected with its associated, *but unchanged,* parity function. This fact justifies the last transition of the rightmost differing double bit, where its associated parity function will not be changed by definition, in a routing when R_3 is needed.

Given a center vertex c, to enumerate those vertices u such that $d_{TQ}(c,u) = i, i \in [2, m+1]$, by Proposition 1, we need to enumerate two groups of such vertices: those vertices containing i double bits which are different from those in c, where R_3 is not applied during the routing process; and those vertices that contain $i-1$ double bits different from those in c, but R_3 is applied exactly once to one of those differing double bits.

For those vertices u, where there exist i different double bits, it is convenient for us to discuss two sub-cases:

- If $u_0 = \overline{c_0}$, then in any of the $\binom{m}{i-1}$ different ways of placing the remaining $i-1$ different double bits, denoted as $u^d_k, k \in [1, i-1]$, among m such double bits, if any of them cannot be corrected via R_2 because of its associated parity function, it will be able to, once u_0, the rightmost differing double bit, is corrected to c_0, which changes the values of all the blocking parity functions to their opposite, respectively. Since for each such a bit, any of the three differing bits can be placed, the number of vertices that are i distance apart, falling into this case, is simply $3^{i-1}\binom{m}{i-1}$.
- Otherwise, $u_0 = c_0$. We use $f(l,k,m,c)$ to denote *the total number of vertices, u, which contains l double bits in the range $[k,m]$, different from their corresponding double bits in c, and no R_3 is ever applied to any of them.* It is clear that the number of vertices that we try to enumerate that fall into this category is simply $f(i,1,m,c)$.

  Since we could only put in at least 0, and at most $m+1$, such double bits in segment $[k,m]$, it is clear that $l \in [0, m+1]$. Moreover, for any k, $f(0,k,m,c) = 1$, since there is exactly one way not to place any differing double bit(s) in a range: simply don't do it.

  For $l \geq 1$, we further categorize this quantity $f(l,k,m,c)$ of all such vertices u, by the position of its rightmost differing double bit, $u_j = u^d_l$. Clearly, $j \geq k$, and since $m-j+1 \geq l$, we have $j \leq m-l+1$. Apparently, this latter quantity depends on c_j, the double bit in c that u^d_l differs, which determines the possible values for u^d_l, as well as the parity function associated with $u^d_l, P_j(u)$, which determines the number of neighbors adjacent to u in terms of any of such possible replacement values. On the other hand, since $u^d_l = u_j$ is the rightmost differing double bit, all the double bits to the right of u_j, including u_0, are equal to their corresponding double bits in c, hence, $P_j(u) = P_j(c)$.

  If we use $f_1(l,j,m,c_j,P_j(c))$ to denote the *total number of vertices that contain l differing double bits in the range $[j,m]$, with no R_3 ever being applied, and the rightmost differing double bit sits in position j*, we have that, for $l \geq 1$,

$$f(l,k,m,c) = \sum_{j=k}^{m-l+1} f_1(l,j,m,c_j,P_j(c)), \qquad (1)$$

  We now derive $f_1(l,j,m,c_j,P_j(c))$ in cases. Let $c_j = c_{2j}c_{2j-1}$.

- If $P_j(c) = 0$, out of three possible values for $u_l^d = u_j$, $\overline{c_{2j}}c_{2j-1}, c_{2j}\overline{c_{2j-1}}$ and $\overline{c_{2j}c_{2j-1}}$, by definition, both $\overline{c_{2j}}c_{2j-1}$ and $\overline{c_{2j}c_{2j-1}}$ can be directly corrected to c_j by applying R_2, but $c_{2j}\overline{c_{2j-1}}$ needs an application of R_3, thus not allowed in this case.

  If $u_l^d = \overline{c_{2j}}c_{2j-1}$, if any of the preceding double bits cannot be corrected because of its associated parity function, when u_l^d corrects to $c_{2j}c_{2j-1}$, the values of all such blocking parity functions will be changed to their respective opposites, thus allowing their associated differing double bits to be corrected by applying R_2. As a result, this value of $\overline{c_{2j}}c_{2j-1}$ will allow any of the three values to be placed in the remaining $l-1$ differing double bits, which can be arranged in any of the $\binom{m-j}{l-1}$ ways. In other words, this case leads to $3^{l-1}\binom{m-j}{l-1}$ vertices.

  On the other hand, if $u_l^d = \overline{c_{2j}c_{2j-1}}$, its correction to $c_{2j}c_{2j-1}$ does not change the value of any of the blocking parity functions, thus, we still need to place the remaining $l-1$ differing double bits in the range $[j+1, m]$, when no R_3 is to be applied, which, by definition, leads to $f(l-1, j+1, m, c)$ vertices. We have to point out here that, although u_j is to contain a different value of $\overline{c_{2j}c_{2j-1}}$ in this latter case, its parity stays the same as that of c_j, thus having no impact on the parity function values associated to the preceding double bits. As a result, the validity of Eq. 1 still holds when applied to $f(l-1, j+1, m, c)$.
- Otherwise, $P_j(c) = 1$. Then, out of the three possible values for $u_l^d = u_j$, both $\overline{c_{2j}}c_{2j-1}$ and $c_{2j}\overline{c_{2j-1}}$ can be corrected to c_j without applying R_3, while $\overline{c_{2j}c_{2j-1}}$ can not be. It is also easy to see that, when $u_l^d = \overline{c_{2j}}c_{2j-1}$ or $c_{2j}\overline{c_{2j-1}}$, if any of the preceding differing double bits cannot be corrected because of its associated parity function, once u_l^d is corrected, all the blocked correction of all such bits can proceed. Hence, in this case, we would end up with $2 \cdot 3^{l-1}\binom{m-j}{l-1}$ vertices.

Since the above result is independent of the actual values of c_j, we rewrite the function f_1 as, for $l \geq 1$,

$$f_1(l, j, m, P_j(c)) = \begin{cases} 3^{l-1}\binom{m-j}{l-1} + f(l-1, j+1, m, c), & P_j(c) = 0; \\ 2 \cdot 3^{l-1}\binom{m-j}{l-1}, & P_j(c) = 1. \end{cases} \tag{2}$$

We now consider the case that a vertex u contains $i-1$ differing double bits from the center c, in the range $[k, m]$, and the rule R_3 is applied exactly once. We refer to the number of such vertices as $g(i, 1, m, c)$. In general, $g(l, k, m, c), k \in [1, m-l+2]$, stands for *the total number of vertices, u, which contains $l-1$ double bits in the range $[k, m]$, different from their corresponding double bits in c, and R_3 is applied exactly once to one of these differing double bits during the routing of a message from u to c.* Since we can put in at least 0, and at most m such differing double bits by Proposition 1, it is clear that g is defined in the range of $l \in [1, m]$, and $g(1, k, m, c) = 0$, since the application of R_3 needs at least one differing double bit(s).

For $l \geq 2$, the same argument as we applied for the previous case leads to the following equation:

$$g(l,k,m,c) = \sum_{j=k}^{m-l+2} g_1(l,j,m,c_j,P_j(c)),$$

where

$$g_1(l,j,m,P_j(c)) = \begin{cases} 3^{l-2}\binom{m-j}{l-2} + g(l-1,j+1,m,c), & P_j(c)=0 \\ 3^{l-2}\binom{m-j}{l-2}, & P_j(c)=1. \end{cases} \tag{3}$$

Therefore, we have derived the following general result for the surface area in a twisted cube, $TQ_d, d \geq 1$, and odd, for any given center c.

Theorem 1. *Let $c \in TQ_d$, $d \geq 1$ and odd, and let $m = \frac{d-1}{2}$. For all $i \in [2, m+1]$,*

$$B_{TQ_d,c}(i) = 3^{i-1}\binom{m}{i-1} + f(i,1,m,c) + g(i,1,m,c).$$

In the above, for $k \in [1, m-l+1]$,

$$f(l,k,m,c) = \begin{cases} 1 & l=0 \\ \sum_{j=k}^{m-l+1} f_1(l,j,m,P_j(c)) & l \in [1,m+1]; \end{cases}$$

and, for $k \in [2, m-l+2]$,

$$g(l,k,m,c) = \begin{cases} 0 & l=1 \\ \sum_{j=k}^{m-i+2} g_1(l,j,m,P_j(c)) & l \in [2,m]; \end{cases}$$

where f_1 and g_1 are given in Eqs. 2, and 3.

It is straightforward to convert the above general result into a computer program, either of a recursive or a dynamic programming nature, which leads to the following sample data for $B_{TQ_{17},c}(i), i \in [0,9]$, where $c = 01\,01\,01\,01\,11\,10\,01\,10\,1$: $(1, 17, 186, 1273, 5614, 16403, 31726, 39159, 27945, 8748)$. It agrees with the sequence obtained through a direct BFS search. We also note that this sequence, and all the other sample data that we have generated for $B_{TQ_d,c}(i), i \in [0, m+1]$, are not included in the On-line Encyclopedia of Integer Sequences [16].

Although a twisted cube structure is not vertex symmetric, it does demonstrate some symmetry as Abraham *et al* observed in [1], which we now discuss. Let $c_{k_m \cdots k_1} = \{c | \forall j \in [1,m], P_j(c) = k_j\}$. For example, for $d = 5$, i.e., $m = 2$, $c_0 = c_{00} = \{{**}000, {**}110\}, c_1 = c_{01} = \{{**}011, {**}101\}, c_2 = c_{10} = \{{**}010, {**}100\}$ and $c_3 = c_{11} = \{{**}001, {**}111\}$, where '*' refers to either 0 or 1. It is clear that, for TQ_d, there are 2^m such classes, and each of which contains $2^{d-m} = 2^{m+1}$ vertices.

Since the above general result, for a given center $c = c_m c_{m-1} \cdots c_0$, as presented in Theorem 1, does not depend on the values of the double bits c_j, but its associated parity functions, $P_j(c), j \in [1,m]$, we immediately have the following symmetry result.

Corollary 1. *Let $c, e \in c_k, k \in [0, 2^m - 1]$, for all $i \in [0, m+1]$, $B_{TQ_d,c}(i) = B_{TQ_d,e}(i)$.*

Thus, for each of the 2^m *symmetric families,* $c_k, k \in [0, 2^m - 1]$, all of its members, as defined above, share the same surface area, as Abraham *et al* suggested in [1].

Moreover, from Eq. 2, the following is the only case of f_1 when $P_m(c)$, the parity function associated with u_m, plays a role:

$$f_1(l, m, m, P_m(c)) = \begin{cases} 3^{l-1}\binom{0}{l-1} + f(l-1, m+1, m, c), & P_m(c) = 0; \\ 2 \cdot 3^{l-1}\binom{0}{l-1}, & P_m(c) = 1. \end{cases}$$

But, the term $f(l-1, m+1, m, c)$, by definition, is to place $l-1$ differing double bits into an empty segment $[m+1, m]$, which is impossible to do, except when $l = 1$, which then leads to exactly two choices in both cases: In fact, when $P_m(c) = 0$, u_1^d will take either $\overline{c_{2m}}c_{2m-1}$ or $\overline{c_{2m}c_{2m-1}}$; and when $P_m(c) = 1$, it takes either $\overline{c_{2m}}c_{2m-1}$ or $c_{2m}\overline{c_{2m-1}}$.

Similarly, from Eq. 3, the following is the only case of g_1 when $P_m(c)$ appears:

$$g_1(l, m, m, P_m(c)) = \begin{cases} 3^{l-2}\binom{0}{l-2} + g(l-1, m+1, m, c), & P_m(c) = 0 \\ 3^{l-2}\binom{0}{l-2}, & P_m(c) = 1. \end{cases}$$

Again, the term $g(l-1, m+1, m, c)$, by definition, is to place $l-2$ remaining differing double bits into an empty segment $[m+1, m]$, which again cannot be done except when $l-2 = 0$, i.e., $l = 2$, when this term equals 1. When $P_m(c) = 0$, $u_1^d = c_{2m}\overline{c_{2m-1}}$, and $u_1^d = \overline{c_{2m}c_{2m-1}}$, when $P_m(c) = 1$. Thus, there exists exactly one arrangement for both cases.

Hence, the value of $P_m(c)$ is actually irrelevant in evaluating both $f(i, 1, m, c)$ and $g(i, 1, m, c)$. This observation leads to the following result, which Abraham *et al* also suggested in [1].

Corollary 2. *Let $c \in c_k, k \in [0, 2^{m-1} - 1], e \in c_{2^{m-1}+k}$, for all $i \in [0, m+1]$, $B_{TQ_d,c}(i) = B_{TQ_d,e}(i)$.*

Corollaries 1 and 2, combined together, tell us that there are exactly 2^{m-1} symmetric families in TQ_d, such that the 2^{m+2} vertices belonging to each family share the same surface area.

4 Surface Areas at Specific Centers for the Twisted Cube

The result that we have derived in Theorem 1 is expressed as a recurrence expression, which is sometimes not practically desirable. As a contrast, a closed-form solution, with its length independent of the involved parameters, can be calculated in constant time, and the derivation process itself is also often mathematically challenging and rewarding.

We now present closed-from expressions of the surface areas for some specific centers in TQ_d, where $d = 2m + 1, m \geq 0$.

Corollary 3. *For all* $i \in \left[2, \frac{d+1}{2}\right]$, *let* $c_0 = 0^d$,

$$B_{TQ_d,c_0}(i) = \frac{3^i - 1}{2}\binom{m}{i-1} + \frac{3^i + 1}{2}\binom{m}{i}.$$

Proof: By Theorem 1, for all $i \geq 2$,

$$B_{TQ_d,c_0}(i) = 3^{i-1}\binom{m}{i-1} + f(i,1,m,c_0) + g(i,1,m,c_0).$$

Since for all $j \in [1,m], P_j(c_0) = 0$, successive unfolding of $f(i,1,m,c_0)$ leads to the following: for $k \in [0, i-1]$, as the first argument of the f term is defined for non-negative values only,

$$\begin{aligned} f(i,1,m,c_0) = {} & 3^{i-1}\sum_{j=1}^{m-i+1}\binom{m-j}{i-1} + 3^{i-2}\sum_{j=1}^{m-i+1}\sum_{j_1=1}^{m-i+2}\binom{m-j_1}{i-2} + \cdots \\ & + 3^{i-k-1}\sum_{j=1}^{m-i+1}\sum_{j_1=1}^{m-i+2}\cdots\sum_{j_k=j_{k-1}+1}^{m-i+(k+1))}\binom{m-j_k}{i-(k+1)} \\ & + \sum_{j=1}^{m-i+1}\sum_{j_1=1}^{m-i+2}\cdots\sum_{j_k=j_{k-1}+1}^{m-i+(k+1))} f(i-(k+1), j_k+1, m). \end{aligned}$$

Although the above is pretty intimidating, it turns out that, for all $k \in [0, i-1]$, we have, taking $j_0 = j$,

$$\sum_{j=1}^{m-i+1}\sum_{j_1=1}^{m-i+2}\cdots\sum_{j_k=j_{k-1}+1}^{m-i+(k+1))}\binom{m-j_k}{i-(k+1)} = \binom{m}{i}.$$

This result can be proved by induction, with its base case, when $k = 0$, being the following well-known binomial identity [9, Eq. 5.10]:

$$\sum_{j=1}^{m-i+1}\binom{m-j}{i-1} = \binom{m}{i}.$$

On the other hand, it also has a clear combinatorial interpretation: to place i symbols in m boxes, $m \geq i \geq 0$, we label those boxes with 1 through m, and, after fixing the first k boxes within their respective range, we now use label j_k to choose one more box to place the next symbol. Clearly, $j_k \geq j_{k-1} + 1$, and once j_k is fixed, we still need to place $i-(k+1)$ symbols in the remaining $m - j_k$ boxes, which leads to $j_k \leq m - i + (i+1)$, and the total arrangements of such placements is simply $\binom{m-j_k}{i-(k+1)}$. This process may continue until $k = i-1$, where there is nothing left for us to place in a box.

Note that the above holds particularly for $k = i-1$, when the lower index of the binomial coefficient becomes 0, i.e., the coefficient itself turns to 1, and the fact that $f(l, k, m, c) = 1$ when $l = 0$, we have

$$f(i, 1, m, c_0) = \left[\sum_{j=0}^{i-1} 3^j + 1\right] \binom{m}{i} = \frac{3^i + 1}{2} \binom{m}{i}.$$

In a similar process, since $g(1, k, m, c) = 0$, for all k, we have

$$g(i, 1, m, c_0) = \left[\sum_{j=0}^{i-2} 3^j\right] \binom{m}{i-1} = \frac{3^{i-1} - 1}{2} \binom{m}{i-1}.$$

The result then follows. □

At the other end of the spectrum, when no unfolding is needed, we have the easiest case.

Corollary 4. *For all* $i \in \left[2, \frac{d+1}{2}\right]$, *let* $c_1 = 0^d1$.

$$B_{TQ_d,c_1}(i) = 4 \cdot 3^{i-2} \binom{m}{i-1} + 2 \cdot 3^{i-1} \binom{m}{i}.$$

We end this section by presenting another surface area result for TQ_d, with radius $i \in [2, m+1]$, centered at $0^{d-3}010$.

Corollary 5. *For all* $i \in \left[2, \frac{d+1}{2}\right]$, *let* $c_2 = 0^{d-3}010$.

$$B_{TQ_d,c_2}(i) = 3^{i-1} \binom{m}{i-1} + 2 \cdot 3^{i-1} \binom{m-1}{i} + 6 \cdot 3^{i-2} \binom{m-1}{i-1} + \left[3^{i-2} + [i > 2]\, 3^{i-3}\right] \binom{m-1}{i-2},$$

where $[i > 2] = 1$ *if* $i > 2$, *and 0 otherwise.*

Proof: For the given center $C_2 = 0^{d-3}010$, it is clear that $P_1(C_2) = 0$, and for all $j \in [2, m], P_j(C_2) = 1$. Thus, we have

$$\begin{aligned} f(i, 1, m, C_2) &= \sum_{j=1}^{m-i+1} f_1(i, j, m, P_j(C_2)) = f_1(i, 1, m, 0) + \sum_{j=2}^{m-i+1} f_1(i, j, m, 1) \\ &= f_1(i, 1, m, 0) + 2 \cdot 3^{i-1} \sum_{j=2}^{m-i+1} \binom{m-j}{i-1}. \end{aligned}$$

Since

$$\sum_{j=2}^{m-i+1} \binom{m-j}{i-1} = \sum_{j=i-1}^{m-1} \binom{j}{i-1} = \binom{m-1}{i},$$

and

$$f_1(i,1,m,0) = 3^{i-1}\binom{m-1}{i-1} + f(i-1,2,m) = 3^{i-1}\binom{m-1}{i-1} + \sum_{j=2}^{m-i+2} f_1(i-1,j,m,1)$$

$$= 3^{i-1}\binom{m-1}{i-1} + 2\cdot 3^{i-2}\sum_{j=2}^{m-i+2}\binom{m-j}{i-2} = 3^{i-1}\binom{m-1}{i-1} + 2\cdot 3^{i-2}\binom{m-1}{i-1}.$$

We have

$$f(i,1,m,C_2) = 2\cdot 3^{i-1}\binom{m-1}{i} + 5\cdot 3^{i-2}\binom{m-1}{i-1}.$$

Similarly,

$$g(i,1,m,C_2) = 3^{i-2}\binom{m-1}{i-1} + \left(3^{i-2} + [i>2]3^{i-3}\right)\binom{m-1}{i-2}.$$

The result then follows. □

It is clear that, for any given center, c, we can follow the approach as demonstrated in this section to derive an explicit form of the surface area result for the twisted cube relative to c.

5 Comparison of the Twisted Cube and Other Variants

We now apply some of the results that we have derived in the previous sections to present a refined analysis of the average distance of TQ_d in terms of some exemplary centers.

We first present, in Table 1, the surface areas for TQ_{19}, centered at four different centers: c_0, c_1, c_2, as defined in the previous section, and $c_{ABCD} =$ 00 01 01 01 01 11 10 01 10 1, with its hexadecimal equivalent being $ABCD$.

Table 1. Surface area of TQ_{19} for different centers

	0	1	2	3	4	5	6	7	8	9	10
c_0	1	19	216	1644	8,526	30,618	76,524	131,196	147,609	98,411	29,524
c_1	1	19	252	1,944	9,828	34,020	81,648	134,136	144,342	91,854	26,244
c_2	1	19	244	1,868	9,408	32,634	79,002	131,868	145,314	95,499	28,431
c_{ABCD}	1	19	234	1,829	9,435	33,248	80,936	134,337	145,422	92,583	26,244

Although the above four cases represent only four out of 256 symmetric families of TQ_{19}, it is clear that the differences in the distance distribution of the nodes are not small across the various cases, as suggested in [1].

We now address the average distance issue. The following result is given in [9, Eq. 5.18], for any real r, and integer m,

$$\sum_{k\le m}\binom{r}{k}\left(\frac{r}{2}-k\right) = \frac{m+1}{2}\binom{r}{m+1}.$$

We then immediately obtain that the average distance of the vertex symmetric hypercube with d dimensions, denoted as HQ_d, is $\frac{d}{2}, d \geq 1$.

We have also obtained the following surface area result [5] for another variant of the hypercube, the vertex symmetric augmented cube [6], denoted as AQ_d : $B_{AQ_d}(0) = 1$, and for $i \in [1, D(AQ_d)]$,

$$B_{AQ_d}(i) = 2^{i-1}\left[\binom{d-i+1}{i} + \binom{d-i}{i}\right],$$

which we use to derive an average distance expression for this structure, as discussed in Section 1.

We now apply Theorem 1 to calculate the average distances of $TQ_d, d \in [1, 19]$, *relative to different centers,* which is then compared, in Table 2, with HQ_d and AQ_d, where the data for $TQ_d(c_{ABCD})$ are not available for $d \in [1, 15]$ since the index for this center contains 17 bits, hence its average distance can not be calculated for anything less.

Table 2. Average distance comparison among TQ_d, AQ_d and HQ_d

d	$TQ_d\,(c_0)$	$TQ_d\,(c_1)$	$TQ_d\,(c_2)$	$TQ_d(c_{ABCD})$	AQ_d	HQ_d
1	0.5	0.5	0.5	N/A	0.5	0.5
3	1.38	1.38	1.38	N/A	0.88	1.5
5	2.19	2.16	2.19	N/A	1.66	2.5
7	2.97	2.91	2.95	N/A	2.40	3.5
9	3.73	3.67	3.71	N/A	3.10	4.5
11	4.49	4.42	4.46	N/A	3.77	5.5
13	5.27	5.17	5.21	N/A	4.44	6.5
15	6.00	5.92	5.96	N/A	5.11	7.5
17	6.74	6.67	6.71	6.68	5.77	8.5
19	7.50	7.42	7.46	7.43	6.44	9.5

This set of data shows that the average distances of the twisted cube, relative to various centers, are fairly close to each other, in all sizes, despite having quite different surface areas among themselves.

6 Concluding Remarks

In this paper, we derived a general surface area result for the asymmetric twisted cube structure for arbitrary but fixed centers, which enables us to accurately characterize the average distance behavior of this asymmetric structure, relative to any given center(s).

References

1. Abraham, S., Padmanabhan, K.: The twisted cube topology for multiprocessor: A study in network asymmetry. J. Parallel Distrib. Comput. 13, 104–110 (1991)
2. Bhaskar, R., Cheng, E., Liang, M., Pandey, S., Wang, K.: Matching preclusion and conditional matching preclusion problems for twisted cubes. Congressus Numerantium 205, 175–185 (2010)
3. Chang, C.P., Wang, J.N., Hsu, L.H.: Topological properties of twisted cubes. Inform. Sci. 113, 147–167 (1999)
4. Cheng, E., Qiu, K., Shen, Z.: On the surface areas and average distances of meshes and tori. Parallel Processing Letters 21(1), 61–75 (2011)
5. Cheng, E., Qiu, K., Shen, Z.: On the surface area of the augmented cubes (2010) (manuscript)
6. Choudum, A., Sunitha, V.: Augmented cubes. Networks 40, 71–84 (2002)
7. Fan, J., Zhang, S., Jia, X., Zhang, G.: A fault-free unicast algorithm in twisted cubes with the restricted faulty node set. In: Proc. 2009 15th International Conference on Parallel and Distributed Systems (ICPADS 2009), December 8-11, pp. 316–323. IEEE Computer Society, Shenzhen (2009)
8. Fertin, G., Raspaud, A.: k-Neighbourhood broadcasting. In: 8th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2001), pp. 133–146 (2001)
9. Graham, R., Knuth, D., Patashnik, O.: Concrete Mathematics. Addison-Wesley, Reading (1989)
10. Hilbers, P., Koopman, M., van de Snepscheut, J.: The twisted cube. In: Treleaven, P.C., Nijman, A.J., de Bakker, J.W. (eds.) PARLE 1987. LNCS, vol. 258, pp. 152–159. Springer, Heidelberg (1987)
11. Huang, W., Tan, J., Hung, C., Hsu, L.: Fault-tolerant Hamiltonicity of twisted cubes. J. Parallel Distrib. Comput. 62, 591–604 (2002)
12. Imani, N., Sarbazi-Azad, H., Akl, S.G.: Some topological properties of star graphs: The surface area and volume. Discrete Mathematics 309(3), 560–569 (2009)
13. Portier, F., Vaughan, T.: Whitney numbers of the second kind for the star poset. Europ. J. Combinatorics 11, 277–288 (1990)
14. Sarbazi-Azad, H., Khonsari, A., Ould-Khaoua, M.: On the topological properties of grid-based interconnection networks: surface area and volume of radial spheres. Computer Journal doi:10.1093/comjnl/bxq011
15. Shen, Z., Qiu, K., Cheng, E.: On the surface area of the (n,k)-star graph. Theoretical Computer Science 410(52), 5481–5490 (2009)
16. Sloane, N.: The On-Line Encyclopedia of Integer Sequences, `http://oeis.org/`
17. Wang, L., Subrammanian, S., Latifi, S., Srimani, P.K.: Distance distribution of nodes in star graphs. Applied mathematics Letters 19(8), 780–784 (2006)
18. Yang, M., Li, T., Tan, J., Hsu, L.: On embedding cycles into faulty twisted cubes. Inform. Sci. 176(6), 676–690 (2006)

Tractable Feedback Vertex Sets in Restricted Bipartite Graphs[⋆]

Wei Jiang[1], Tian Liu[1], and Ke Xu[2]

[1] Key Laboratory of High Confidence Software Technologies, Ministry of Education, Institute of Software, School of Electronic Engineering and Computer Science, Peking University, Beijing 100871, China
lt@pku.edu.cn

[2] National Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
kexu@nlsde.buaa.edu.cn

Abstract. A feedback vertex set (FVS) is a subset of vertices whose removal renders the remaining graph a forest. We show that finding the minimum FVS is tractable in the so-called triad convex bipartite graphs.

1 Introduction

A *feedback vertex set* (FVS), sometimes also called *loop cutset*, in a graph is a subset of vertices whose removal renders the remaining graph a forest. In weighted graphs, the weight of a FVS is the summation of the weight of each vertex in the FVS, and the weight of each vertex is a positive integer. For non-weighted graphs, we just assume that each vertex has a weight one. Finding the minimum FVS (MFVS) even in non-weighted graphs is a classical $\mathcal{NP}$-complete problem. In fact, it was among the twenty-one $\mathcal{NP}$-complete problems in Karp's list [19]. MFVS has applications in deadlock prevention and recovery in operating systems [28], information security [15], VLSI chip design [10], artificial intelligence [1,30], etc.. Many algorithms have been developed for MFVS, which are approximate [3,1,4,5,27,33], randomized [2], parameterized [8,9,6], exact [12], polynomial in restricted graphs [22,24,21,29,14,20], to enumerate and count the number of MFVS [11], and to estimate the size of MFVS [26]. In this paper, we show a tractable result about MFVS in some restricted bipartite graphs. Namely, we show that MFVS is polynomial in restricted bipartite graphs which are called triad convex.

A Bipartite graph $G = (A, B, E)$ is called *tree convex*, if there is a tree $T = (A, F)$, such that for all vertex b in B, the subset $N(b) = \{a \in A | (a, b) \in E\}$ (i.e. the neighborhood of b in A) is a connected subtree in T. When T is a star (i.e. a bipartite complete graph $K_{1,|A|-1}$), G is called *star convex*. When T is a path, G is called *path convex* or just *convex*. When T is a triad (i.e. three paths with a common end), G is called *triad convex*. It was known that finding

[⋆] To whom the correspondence should be addressed: Tian Liu (lt@pku.edu.cn) and Ke Xu (kexu@nlsde.buaa.edu.cn).

MFVS in non-weighted bipartite graphs is also $\mathcal{NP}$-hard [31], but tractable in weighted convex bipartite graphs and cocomparable graphs [22], interval graphs [24], circle graphs [14], mesh and butterfly [7,25], hypercube [13], star graph [29], rotator graph [16,20], and shuffle-based interconnection networks [21], etc..

Recently, we generalized the notion of convex bipartite graphs to the notion of tree convex bipartite graphs, and showed that finding MFVS in non-weighted star convex bipartite graphs is also $\mathcal{NP}$-hard [17]. Thus the tractability result of MFVS in convex bipartite graphs in [22] is unlikely extensible to all tree convex bipartite graphs. Here in contrast to [17], we show that finding MFVS in weighted triad convex bipartite graphs is tractable. Therefore, the tractability result in [22] on convex bipartite graphs is at least extensible to triad convex bipartite graphs. In summary, the results in [17] and here have refined the complexity classification of MFVS in [31,22] on various restricted bipartite graphs.

This paper is organized as follows. In Section 2, we recall some necessary definitions and notations along with some known results on FVS and tree convex bipartite graphs. In Section 3, we describe the details of an algorithm for finding MFVS in weighted triad convex bipartite graphs, and also show its correctness. In Section 4, we give an explicit polynomial time bound for the algorithm and remark some possible extension and limit of the algorithm. Finally, we discuss some open problems.

2 Definitions

In this section, we recall some definitions and notations along with some known results to be used in subsequent sections about FVS and convex bipartite graphs as in [22] and about tree convex bipartite graphs as in [17].

A Bipartite graph $G = (A, B, E)$ is called *tree convex*, if there is a tree $T = (A, F)$, such that for all vertex b in B, the subset $N(b) = \{a \in A | (a, b) \in E\}$ (i.e. the neighborhood of b in A) is a connected subtree in T [17]. When T is a triad (i.e. three paths with a common end), G is called *triad convex* [17]. When T is a path, G is called *path convex* or just *convex* [23]. For convex bipartite graphs, we can equivalently assume that there is a linear ordering $<$ on A, such that each neighborhood of some b is an interval of A under $<$, see e.g. [22].

Although the notion of tree convex bipartite graphs was recently introduced by us in [17], from the results in [32] we can know that both convex bipartite graphs and tree convex bipartite graphs are recognizable in linear time, and the associated linear ordering $<$ on A in convex bipartite graphs and the associated tree T on A in tree convex bipartite graphs are both constructible in linear time. Thus we can safely assume that they are parts of the inputs.

Assume a convex bipartite graph $G = (A, B, E)$, $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_m\}$, with a linear ordering $a_i < a_j$ for $i < j$ on A. For convenience, we add two more vertices a_0 and a_{n+1} to A [22]. The following notations are also from [22]:

- For a vertex v in a graph, $N(v)$ denotes the set of all vertices adjacent to v.
- For every $b \in B$ in G, let $L[b]$ and $R[b]$ denote the smallest and largest vertices, respectively, of A connected to b. Let $B = \{b_1, \ldots, b_m\}$ with $R[b_i] < R[b_j]$ for $1 \leq i < j \leq m$.
- For $a_i \in A$ where $0 \leq i \leq n+1$, let $A_i = \{a_h : a_h \in A \text{ and } 0 \leq h \leq i\}$ and $B_i = \{b : b \in B \text{ and } R[b] < a_i\}$.
- For $a_i, a_j, a_k \in A$ where $0 \leq i < j < k \leq n+1$, define $B_{i,j} = \{b : b \in B \text{ and } a_i < L[b] \leq R[b] < a_j\}$, and $B_{i,j,k} = \{b : a_i < L[b] \text{ and } a_j \leq R[b] < a_k\}$.
- For $a_i, a_j \in A$ where $i < j$, $M(i,j)$ denotes a MCFS in $A_j + B_j$ containing $\{a_i, a_j\}$(i.e. $a_i, a_j \in M(i,j)$), where a_i and a_j are the largest two vertices in $A \cap M(i,j)$.
- For $a_i, a_j, a_k \in A$ where $0 \leq i < j < k \leq n+1$, $M(i,j,k)$ denotes a MCFS in $A_k + B_k$ containing $\{a_i, a_j, a_k\}$, where a_i, a_j, a_k are the largest three vertices in $A \cap M(i,j,k)$.
- For $a_i, a_j \in A$ and $b_k \in B$ where $0 < i < j < n+1$ and $b_k \in N(a_i) \cap N(a_j)$, $M_b(i,j,k)$ denotes a MCFS in $A_j + B_j + \{b_k\}$ containing $\{a_i, a_j, b_k\}$, where a_i and a_j are the largest two vertices in $A \cap M_b(i,j,k)$.

In [22], it was shown that all the $M(i,j)$, $M(i,j,k)$ and $M_b(i,j,k)$ can be computed in $O(|A|^3 + |A|^2|E|)$ time. A convex bipartite graph with the added vertices a_0 and a_{n+1} is shown in figure 1.

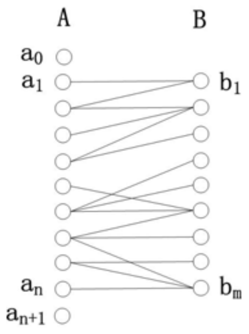


Fig. 1. A convex bipartite graph

Assume a triad convex bipartite graph $G = (A, B, E)$ with a triad $T = (A, F)$ defined on A, such that every $N(b)$ is a subtree of T for $b \in B$ [17]. To be specific, let $A = \{a_0\} \cup A_1 \cup A_2 \cup A_3$, such that for $1 \leq i \leq 3$, $A_i = \{a_{i,1}, \ldots, a_{i,n_i}\}$ and $a_0 \to a_{i,1} \to \ldots \to a_{i,n_i}$ are three paths with a common end a_0. The vertices of a triad convex bipartite graph (without edges) are shown in figure 2.

We can assume that the graphs are weighted with positive integers on their vertices. For non-weighted graphs, these weights are assumed to be one. Recall that removing a FVS renders the remaining graph a forest, and the weight of a FVS is the sum of weights of vertices in the FVS. Thus finding a minimum FVS

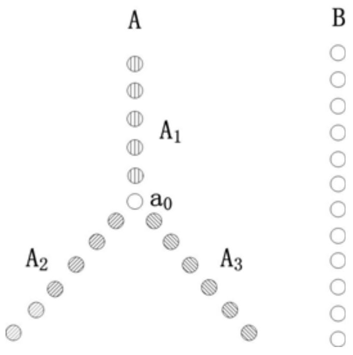


Fig. 2. A triad convex bipartite graph (without edges)

(MFVS) in a graph is equivalent to finding a maximum cycle-free set (MCFS) in the same graph [22]. Below we will give an algorithm to find a MCFS instead of a MFVS in weighted triad convex bipartite graphs as in [22].

3 The Algorithm

In this section, we will describe an algorithm for MFVS in weighted triad convex bipartite graphs. This algorithm is a natural extension of the known algorithm in [22]. Along with the description, we will also show the correctness of the algorithm.

Let U be the MCFS of G, we will consider the following two cases: $a_0 \notin U$ and $a_0 \in U$.

Case 1**:** $a_0 \notin U$.
Let $a_{1,i}, a_{2,j}, a_{3,k}$ be the nearest vertices to a_0 in U on each path A_1, A_2, A_3 respectively. Some of them may not exist at all, that is, the corresponding whole path may not intersect with U. we can divide A into four parts S_0, S_1, S_2, S_3 as follows:
$S_0 = \{a_0, a_{1,1}, \ldots, a_{1,i-1}, a_{2,1}, \ldots, a_{2,j-1}, a_{3,1}, \ldots, a_{3,k-1}\}$,
$S_1 = \{a_{1,i}, \ldots, a_{1,n_1}\}$,
$S_2 = \{a_{2,j}, \ldots, a_{2,n_2}\}$,
$S_3 = \{a_{3,k}, \ldots, a_{3,n_3}\}$.
An example is shown in figure 3. Note that if some vertices in $\{a_{1,i}, a_{2,j}, a_{3,k}\}$ do not exist, then the corresponding sets in $\{S_1, S_2, S_3\}$ will be empty, however this will cause no harms to our algorithm but only helps.

We can put each $v \in B$ into one of the following six sets $B_1, \ldots, B_6$ according to the situation of v's neighborhood:
$B_1 = \{v \mid a_{1,i}, a_{2,j} \in N(v), a_{3,k} \notin N(v)\}$.
$B_2 = \{v \mid a_{1,i}, a_{3,k} \in N(v), a_{2,j} \notin N(v)\}$.
$B_3 = \{v \mid a_{2,j}, a_{3,k} \in N(v), a_{1,i} \notin N(v)\}$.

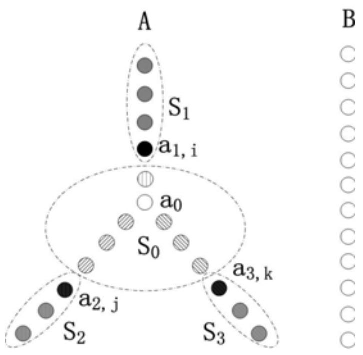


Fig. 3. Divide A into four parts

$B_4 = \{v \mid a_{1,i}, a_{2,j}, a_{3,k} \in N(v)\}$.
$B_5 = \{v \mid a_{1,i}, a_{2,j}, a_{3,k} \notin N(v), N(v) \subseteq S_0\}$.
$B_6 = \{v \mid |\{a_{1,i}, a_{2,j}, a_{3,k}\} \cap N(v)| \le 1,$ and $N(v) \cap (S_1 \cup S_2 \cup S_3) \neq \emptyset\}$.
The following observations are easily seen.

(1) At most one vertex from B_1 is selected into U (similarly for B_2,B_3,B_4). Otherwise, if both $b_p, b_q \in U$ are from B_1, then U will contain a cycle $b_p \to a_{1,i} \to b_q \to a_{2,j} \to b_p$. Let b_1, b_2, b_3, b_4 be the vertices possibly selected into U from B_1, B_2, B_3, B_4 respectively. Some of them may not exist, since they are not selected into U at all, see below.

(2) At most two vertices in $\{b_1, b_2, b_3\}$ or at most one vertex in $\{b_4\}$ are available in any MCFS, since $\{b_1, b_2, b_3\}$ will form a cycle $b_1 \to a_{1,i} \to b_2 \to a_{3,k} \to b_3 \to a_{2,j} \to b_1$, and for example $\{b_1, b_4\}$ will form a cycle $b_1 \to a_{1,i} \to b_4 \to a_{2,j} \to b_1$.

(3) All the vertices from B_5 should be in U, since obviously they are not contained in any cycle.

(4) All the edges between B_6 and S_0 are removable, since they do not change the result of MCFS.

Obviously, the selection of $a_{1,i}$,$a_{2,j}$,$a_{3,k}$ is done in at most $O(|A|^3)$ time. According to the above observations (1) and (2), we can enumerate the vertices b_1, b_2, b_3, b_4 in $O(|B|^2)$ time.

After selecting $a_{1,i}$,$a_{2,j}$,$a_{3,k}$ and b_1, b_2, b_3, b_4 as above, by the above observation (1), we can remove all vertices in $B_i \setminus \{b_i\}$ from G. After the removing, we define four induced subgraphs G_0,G_1,G_2,G_3 of G and find MCFS in G_1,G_2,G_3 as follows (also see figure 4). For any bipartite graph $G = (A, B, E)$ and any subsets $C \subseteq A$ and $D \subseteq B$, denote by $G[C, D]$ the subgraph of G induced by C and D, which is the bipartite graph $(C, D, E \cap (C\&D))$. Here $C\&D = \{(c, d)|c \in C$ and $d \in D\}$ is the set of all unordered pairs of vertices in C and D. Let $N(C) = \{b|b \in B$ and b has a neighbor $c \in C\}$. Then the graphs G_0, G_1, G_2, G_3 are defined as $G[S_0, N(S_0)], G[S_1, N(S_1)], G[S_2, N(S_2)], G[S_3, N(S_3)]$ respectively. An example of the vertex sets of subgraphs G_0,G_1,G_2,G_3 is shown in figure 4.

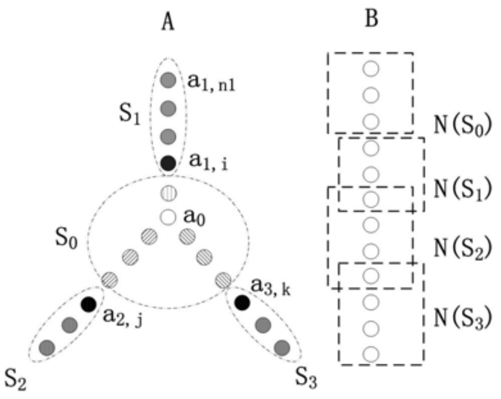


Fig. 4. The vertex sets (without edges) of subgraphs G_0, G_1, G_2, G_3

Let U_1,U_2,U_3 be MCFS of G_1,G_2,G_3 respectively. It is obvious that $U = U_1 \cup U_2 \cup U_3 \cup B_5$. Let $X_1 = \{v | v \in \{b_1, b_2, b_4\}, v$ is actually selected into $U\}$. By above observation (2), we know that $|X_1| \leq 2$. Below we will show how to find the MCFS U_1 in G_1 conditioned on X_1 (we can find U_2, U_3 in G_2, G_3 similarly).

Assume that $|S_1| = n_1'$. For convenience, we add two isolated vertices $a_0', a_{n_1'+1}'$ into S_1 and make an ordering on $N(S_1)$ as in [22]. For the purposes to deal with U_2, U_3 in a similar way to U_1 and to use the algorithm in [22], we rename the vertices in S_1 as $a_0', a_1', \cdots, a_{n_1'+1}'$ and rename $S_1, N(S_1), G_1, U_1, X_1, n_1'$ as A, B, H, U, X, n respectively. Also for convenience, we will omit the primes in a_i''s and b_j''s, and use a_i's and b_j's instead in the following lemma. This renaming procedure is partially depicted in figure 5.

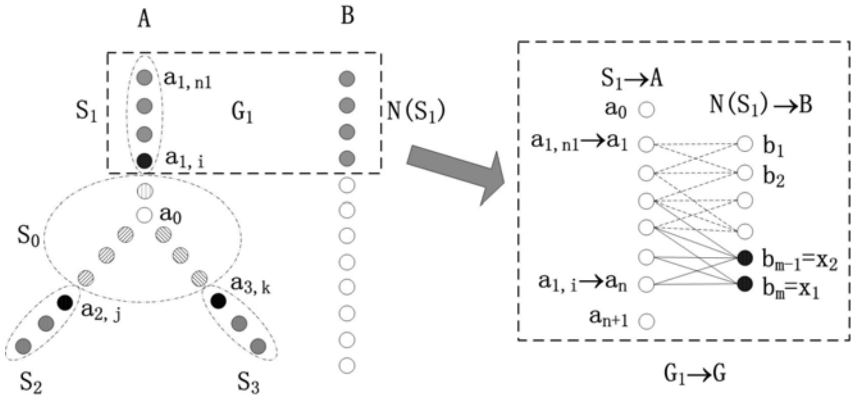


Fig. 5. Renaming vertices and finding U_1 in G_1

Lemma 1. *The MCFS U of H is computed in the following three cases accordingly:*

1. *If $X = \emptyset$, $U = M(n, n+1) \setminus \{a_0, a_{n+1}\}$.*
2. *If $X = \{x_1\}$, U is one of the following candidates, whichever has maximum weight:*
   (a) $M_b(i, n, x_1) \cup B_{i,n,n+1} \setminus \{a_0\}$, for some $i, a_i \in N(x_1)$.
   (b) $M(i, n, n+1) \setminus \{a_0\}$, for some $i, a_i \in A \setminus N(x_1)$.
3. *If $X = \{x_1, x_2\}$ and $|N(x_2) \cap A| \geq |N(x_1) \cap A|$, U is one of the following candidates, whichever has maximum weight:*
   (a) $M_b(i, n, x_2) \cup B_{i,n,n+1} \setminus \{a_0\}$, for some $i, a_i \in (N(x_2) \setminus N(x_1))$.
   (b) $M(i, n, n+1) \setminus \{a_0\}$, for some $i, a_i \in A \setminus N(x_2)$.

Proof. Consider the following three cases.

1. $X = \emptyset$.
   U_1 is obviously $M(n, n+1) \setminus \{a_0, a_{n+1}\}$ according to the definition of $M(i, j)$.
2. $X = \{x_1\}$.
   If the largest three vertices is a_i, a_n, a_{n+1}, both $M_b(i, n, x_1) \cup B_{i,n,n+1} \setminus \{a_0\}$ (when $a_i \in N(x_1)$) and $M(i, n, n+1) \setminus \{a_0\}$ (when $a_i \in A \setminus N(x_1)$) are cycle-free, and one of them is the MCFS of U according to the definitions of $M(i, j, k)$, $M_b(i, j, k)$ and $B_{i,j,k}$.
3. $X = \{x_1, x_2\}$ and $|N(x_2) \cap A| \geq |N(x_1) \cap A|$.
   If the largest three vertices is a_i, a_n, a_{n+1}, $a_i \notin |N(x_1) \cap A|$, otherwise U contains a cycle $x_1 \to a_i \to x_2 \to a_n \to x_1$. Both $M_b(i, n, x_2) \cup B_{i,n,n+1} \setminus \{a_0\}$ (when $a_i \in N(x_2) \setminus N(x_1)$) and $M(i, n, n+1) - \{a_0\}$ (when $a_i \in A \setminus N(x_2)$) are cycle-free, and one of them is the MCFS of U_1 according to the definitions of $M(i, j, k)$, $M_b(i, j, k)$ and $B_{i,j,k}$.

This finishes the proof. □

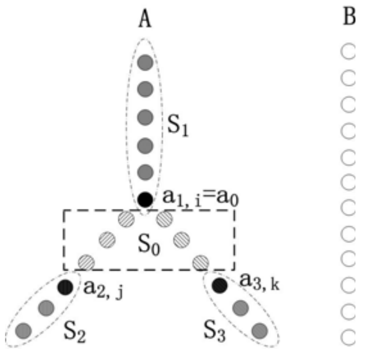


Fig. 6. Set S of Case 2

Case 2: $a_0 \in U$.
Similar to Case 1, let $a_{1,i} = a_0$ and $a_{2,j}, a_{3,k}$ be the nearest vertices to a_0 in U on each path A_2, A_3 respectively. Some of $a_{2,j}, a_{3,k}$ may not exist at all, that is, the corresponding whole path may not intersect with U. We can also divide A into four parts S_0, S_1, S_2, S_3 as follows:

$S_0 = \{a_{2,1}, \ldots, a_{2,j-1}, a_{3,1}, \ldots, a_{3,k-1}\}$,
$S_1 = \{a_0, a_{1,1}, \ldots, a_{1,n_1}\}$,
$S_2 = \{a_{2,j}, \ldots, a_{2,n_2}\}$,
$S_3 = \{a_{3,k}, \ldots, a_{3,n_3}\}$.

An example is shown in figure 6. Also note that some of S_i's maybe empty.

We can put each $v \in B$ into one of the following five sets $B_1, \ldots, B_5$ according to the situation of v's neighborhood:

$B_1 = \{v \mid a_0, a_{2,j} \in N(v), a_{3,k} \notin N(v)\}$.
$B_2 = \{v \mid a_0, a_{3,k} \in N(v), a_{2,j} \notin N(v)\}$.
$B_3 = \{v \mid a_0, a_{1,i}, a_{2,j} \in N(v)\}$.
$B_4 = \{v \mid a_{1,i}, a_{2,j}, a_{3,k} \notin N(v), N(v) \subseteq S_0\}$.
$B_5 = \{v \mid |\{a_0, a_{2,j}, a_{3,k}\} \cap N(v)| \leq 1, \text{ and } N(v) \cap (S_1 \cup S_2 \cup S_3) \neq \emptyset\}$

Then we will immediately get the similar observations and lemmas as in Case 1, so we omit the details here.

The main steps of the above algorithm are summarized as follows.

```
Input: A triad convex bipartite graph $G = (A, B, E)$ with triad $A = \{a_0\} \cup A_1 \cup A_2 \cup A_3$
Output: A MCFS $U$ of $G$
1: $U = \emptyset$.
2: Enumerate $a_{1,i} \in A_1 \cup \{a_0\}$, $a_{2,j} \in A_2$ and $a_{3,k} \in A_3$.
3: Assume that $a_{1,i}$, $a_{2,j}$ and $a_{3,k}$ are the nearest vertices to $a_0$ in $U$ on paths $\{a_0\} \cup A_1, A_2, A_3$ respectively. Divide $A$ into $S_0, S_1, S_2, S_3$, divide $B$ into $B_1, \ldots, B_6$ (when $a_{1,i} \neq a_0$) or $B_1, \ldots, B_5$ (when $a_{1,i} = a_0$) accordingly.
4: Enumerate $b_1 \in B_1, b_2 \in B_2, b_3 \in B_3, b_4 \in B_4$ (when $a_{1,i} \neq a_0$) or $b_1 \in B_1, b_2 \in B_2, b_3 \in B_3$ (when $a_{1,i} = a_0$).
5: Select at most two from $\{b_1, b_2, b_3\}$ or at most one from $\{b_4\}$ into $U$, and remove all other vertices in $B_1, \ldots, B_4$ (when $a_{1,i} \neq a_0$) or
   select at most two from $\{b_1, b_2\}$ or at most one from $\{b_3\}$ into $U$, and remove all other vertices in $B_1, \ldots, B_3$ (when $a_{1,i} = a_0$).
6: Put $B_5$ into $U$ and remove all edges between $S_0$ and $B_6$ (when $a_{1,i} \neq a_0$) or
   put $B_4$ into $U$ and remove all edges between $S_0$ and $B_5$ (when $a_{1,i} = a_0$).
7: Define subgraphs $G_0, G_1, G_2, G_3$ accordingly and find the MCFS $U_1, U_2, U_3$ in $G_1, G_2, G_3$ respectively. Update $U$ by $U_1 \cup U_2 \cup U_3 \cup B_5$ (when $a_{1,i} \neq a_0$) or
   by $U_1 \cup U_2 \cup U_3 \cup B_4$ (when $a_{1,i} = a_0$).
8: If the enumeration in step 4 is unfinished then goto step 4, else if the enumeration in step 2 is unfinished then goto step 2.
```

Algorithm 1. Finding MFVS in triad convex bipartite graphs

4 The Analysis

In this section, we give an explicit polynomial time bound for the above algorithm, and remark some possible extension and limit of the algorithm here.

Theorem 1. *A MCFS in a triad convex bipartite graphs is found in* $O(|A|^3|B|^2 (|A|^3 + |A|^2|E|))$ *time.*

Proof. Enumerate $a_{1,i}, a_{2,j}, a_{3,k}$ is done in $O(|A|^3)$ and enumerate b_1, b_2, b_3, b_4 is done in $O(|B|^2)$. So the total enumeration time is $O(|A|^3|B|^2)$. According to [22], all $M(i, j), M(i, j, k)$ and $M_b(i, j, k)$ are computed in $O(|A|^3+|A|^2|E|)$ time. Therefore, a MCFS in a triad convex bipartite graphs is found in $O(|A|^3|B|^2 (|A|^3 + |A|^2|E|))$ time. □

Remark 1. A triad is a tree which is three paths with a common end. We can also consider any k paths with a common end, where k is a constant. Although we do not have a name for this kind of trees, it is not hard to see that, using a similar algorithm as above, we can find MCFS in these tree convex bipartite graphs in $O(|A|^k|B|^{k-1}(|A|^3+|A|^2|E|))$ time, when the associated tree is k paths with a common end for constant k.

Remark 2. Recently we show that the sum of degrees larger than two in the associated tree T of tree convex bipartite graphs is an important quantity. Denote this quantity as d. When d is bounded, using a similar algorithm as here, the MFVS in these tree convex bipartite graphs is solvable in $O(|A|^d|B|^{d-1}(|A|^3+|A|^2|E|))$ time. When the number of vertices of degrees larger than two is unbounded but each degrees are bounded (thus d is also unbounded), the MFVS in these tree convex bipartite graphs is still $\mathcal{NP}$-hard [18].

5 Open Problem

It is unknown whether there are better algorithms than the algorithm presented here for MFVS on restricted bipartite graphs. Also, a complete classification of the complexity of finding MFVS on every kind of tree convex bipartite graphs is unknown. What we can say is that even for bounded degrees tree T, MFVS is still $\mathcal{NP}$-hard in these tree convex bipartite graphs [18].

Acknowledgments. We thank Professor Kaile Su for his encouragement and support. We also thank the unknown reviewers whose comments are very helpful to improve our presentations. This research was partially supported by the National 973 Program of China (Grant No. 2010CB328103), the National Natural Science Foundation of China (Grant Nos. 60725207 and 60973033), the Project of Knowledge Innovation Program (PKIP) of Chinese Academy of Sciences (Grant No. KJCX2.YW.W10), and was partially done while Tian Liu and Ke Xu were visiting KITPC at the Chinese Academy of Science.

References

1. Bar-Yehuda, R., Geiger, D., Naor, J., Roth, R.: Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. SIAM J. Comput. 27, 942–959 (1998)
2. Becker, A., Bar-Yehuda, R., Geiger, D.: Randomized Algorithms for the Loop Cutset Problem. J. Artif. Intell. Res. 12, 219–234 (2000)
3. Becker, A., Geiger, D.: Optimization of Pearls method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. Artificial Intelligence 83, 167–188 (1996)
4. Cai, M., Deng, X., Zang, W.: An approximation algorithm for feedback vertex sets in tournaments. SIAM J. Comput. 30, 1993–2007 (2001)
5. Cai, M., Deng, X., Zang, W.: A min-max theorem on feedback vertex sets. Math. Oper. Res. 27(2), 361–371 (2002)
6. Cao, Y., Chen, J., Liu, Y.: On Feedback Vertex Set: New Measure and New Structures. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 93–104. Springer, Heidelberg (2010)
7. Caragiannis, I., Kaklamanis, C., Kanellopoulos, P.: New bounds on the size of the feedback vertex set on meshes and butterflies. Inform. Process. Lett. 83(5), 275–280 (2002)
8. Chen, J., Fomin, F., Liu, Y., Lu, S., Villanger, T.: Improved algorithms for feedback vertex set problems. J. Comput. Syst. Sci. 74(7), 1188–1198 (2008)
9. Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A fixed-parameter algorithm for the directed feedback vertex set problem. J. ACM 55(5) (2008)
10. Festa, P., Pardalos, P.M., Resende, M.G.C.: Feedback set problems. In: Du, D.-Z., Pardalos, P.M. (eds.) Handbook of Combinatorial Optimization. Handbook of Combinatorial Optimization, Supplement vol. A, pp. 209–259. Kluwer Academic Publishers, Dordrecht (2000)
11. Fomin, F.V., Gaspers, S., Pyatkin, A., Razgon, I.: On the Minimum Feedback Vertex Set Problem: Exact and Enumeration Algorithms. Algorithmica 52(2), 293–307 (2008)
12. Fomin, F.V., Villanger, Y.: Finding Induced Subgraphs via Minimal Triangulations. In: Proc. of STACS, pp. 383–394 (2010)
13. Focardi, R., Luccio, F.L., Peleg, D.: Feedback vertex set in hypercubes. Inform. Process. Lett. 76(1-2), 1–5 (2000)
14. Gavril, F.: Minimum weight feedback vertex sets in circle graphs. Information Processing Letters 107, 1–6 (2008)
15. Gusfield, D.: A graph theoretic approach to statistical data security. SIAM J. Comput. 17(3), 552–571 (1998)
16. Hsu, C.C., Lin, H.R., Chang, H.C., Lin, K.K.: Feedback Vertex Sets in Rotator Graphs. In: Gavrilova, M.L., Gervasi, O., Kumar, V., Tan, C.J.K., Taniar, D., Laganá, A., Mun, Y., Choo, H. (eds.) ICCSA 2006. LNCS, vol. 3984, pp. 158–164. Springer, Heidelberg (2006)
17. Jiang, W., Liu, T., Ren, T., Xu, K.: Two Hardness Results on Feedback Vertex Sets. In: Zhu, B. (ed.) FAW-AAIM 2011. LNCS, vol. 6681, pp. 233–243. Springer, Heidelberg (2011)
18. Jiang, W., Liu, T., Xu, K.: Feedback Vertex Sets in Restricted Bipartite Graphs, paper in preparation
19. Karp, R.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)

20. Kuo, C., Hsu, C., Lin, H., Lin, K.: An efficient algorithm for minimum feedback vertex sets in rotator graphs. Information Processing Letters 109, 450–453 (2009)
21. Kralovic, R., Ruzicka, P.: Minimum feedback vertex sets in shufflebased interconnection networks. Inform. Process. Lett. 86(4), 191–196 (2003)
22. Liang, Y.D., Chang, M.S.: Minimum feedback vertex sets in cocomparability graphs and convex bipartite graphs. Acta Informatica 34, 337–346 (1997)
23. Lipski Jr., W., Preparata, F.P.: Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. Acta Informatica 15(4), 329–346 (1981)
24. Lu, C., Tang, C.: A linear-time algorithm for the weighted feedback vertex problem on interval graphs. Information Processing Letters 61, 107–111 (1997)
25. Luccio, F.L.: Exact minimum feedback vertex set in meshes and butterflies. Inform. Process. Lett. 66(2), 59–64 (1998)
26. Madelaine, F.R., Stewart, I.A.: Improved upper and lower bounds on the feedback vertex numbers of grids and butterflies. Discrete Math. 308, 4144–4164 (2008)
27. Sasatte, P.: Improved approximation algorithm for the feedback set problem in a bipartite tournament. Operations Research Letters 36, 602–604 (2008)
28. Silberschatz, A., Galvin, P.B., Gagne, G.: Operating Systems Concepts, 6th edn. John Wiley and Sons, Inc., New York (2003)
29. Wang, F.H., Wang, Y.L., Chang, J.M.: Feedback vertex sets in star graphs. Inform. Process. Lett. 89(4), 203–208 (2004)
30. Williams, R., Gomes, C.P., Selman, B.: Backdoors To Typical Case Complexity. In: Proc. of IJCAI, pp. 1173–1178 (2003)
31. Yannakakis, M.: Node-deletion problem on bipartite graphs. SIAM J. Comput. 10, 310–327 (1981)
32. Zhang, Y., Bao, F.S.: A Survey of Tree Convex Sets Test. CoRR abs/0906.0205 (2009)
33. van Zuylen, A.: Linear programming based approximation algorithms for feedback set problems in bipartite tournaments. Theor. Comput. Sci. (in press)

On the Partition of 3-Colorable Graphs

Yang Liu and Qing Wang

University of Texas-Pan American
{yliu,qwang}@cs.panam.edu

Abstract. Exact algorithms have made a little progress for the 3-coloring problem since 1976: improved from $O(1.4422^n n^{O(1)})$ to $O(1.3289^n n^{O(1)})$. The best exact algorithm for the 3-coloring problem is by Beigel and Eppstein, and its analysis is very complicated. In this paper, we study the parameterized 3-coloring problem: partitioning a 3-colorable graph into a bipartite subgraph and an independent set. Taking the size of the bipartite subgraph as the parameter k, we develop the first parameter algorithm of complexity $O^*(1.713^k)$. We use measures other than the given parameter k to achieve better analysis on running time. Such a technique of using novel measures may bring new insight into designing faster algorithms for other NP-hard problems.

1 Introduction

Given a undirected graph G, the *chromatic number* of G is the smallest integer k such that the vertices of G can be colored with k colors and no two adjacent vertices are colored with the same color. The CHROMATIC NUMBER problem is to decide the chromatic number of graphs, and has been intensively studied[11,9,5,3]. The best algorithm for the problem is of time complexity $O(2^n n^{O(1)})$, and requires exponential space $O(2^n n)$[3]. When polynomial space complexity is desired, the best algorithm for this problem has time complexity $O(2.2461^n)$[3]. A well known application of the CHROMATIC NUMBER problem is register allocation[6]. Not only exact solutions but also approximation solutions with good ratios are difficult to find for the CHROMATIC NUMBER problem. Indeed, it is NP-hard to approximate the chromatic number of graphs with ratio n^ϵ for some constant $\epsilon > 0$ unless P=NP[12].

The 3-COLORING problem is a special case of the CHROMATIC NUMBER problem. In the 3-COLORING problem, we are asked to determine whether the chromatic number of graphs is 3 or not. Table 1 shows the history of exact algorithms for the 3-COLORING problem. The table shows that it is faster to find exact solutions for the 3-COLORING problem. Meanwhile, approximation solutions with $O(n^{3/14})$ colors can be found in polynomial time for graphs of chromatic number 3[4]. However, it is NP-hard to color graphs of chromatic number 3 with only 4 colors [10].

The 3-COLORING problem can be viewed from another perspective: partitioning the vertices of a graph into three disjoint independent sets, if such a partition exists. This perspective leads to our study of parameterized complexity of the

Table 1. History of Exact Algorithms for 3-coloring

Authors	Complexity	Year
Lawler	1.4422^n	1976 [11]
Schiemeyer	1.415^n	1994 [14]
Beigel and Eppstein	1.3446^n	1995 [1]
Beigel and Eppstein	1.3289^n	2005 [2]

3-COLORING problem when we take the total number of vertices in two independent sets as a parameter. In this paper, we study the following PARAMETERIZED 3-COLORING problem.

PARAMETERIZED 3-COLORING. Given a graph G, can the vertices of G be partitioned into a bipartite subgraph of at most k vertices and an independent set? Find such a partition if it exists, or report 'NO' otherwise.

Algorithms of complexity $O(f(k)n^{O(1)})$ are *fixed-parameter tractable* from the perspective of parameterized complexity. Fixed-parameter tractable(fpt-) algorithms are often practical and fast for applications with small parameter k, by taking advantage of the small k, while exact algorithms do not take such an advantage. In the paper we develop an fpt-algorithm of time complexity $O^*(1.713^k)$[1] for the PARAMETERIZED 3-COLORING problem.

Compared with the PARAMETERIZED 3-COLORING problem, the (H, C, K)-COLORING problem is a more general problem, which is to map vertices of an input graph G to vertices of another graph H such that (1) if there is an edge uv in G then there is a image edge $f(u)f(v)$ in H, and (2) the number of pre-images of a vertex $x \in C \subseteq V(H)$ is equal to its weight $w(x)$ specified by K. This problem has been studied, and an fpt-algorithm $O^*(2^k|C|^k)$ has been developed for a special case when $V(H) - C$ is an independent set [7]. A first look at these two problems shows that the PARAMETERIZED 3-COLORING problem is just a simple special case of the (H, C, K)-COLORING problem when we take H as a triangle, C as an two vertices of H, and $\sum_{w(x)\in K} w(x) = k$. However, the two problems are slightly different: the PARAMETERIZED 3-COLORING problem is concerned only about the total vertices in the bipartite subgraph without caring about how the vertices distributed in the bipartite subgraph, while the (H, C, K)-COLORING problem requires exact number of pre-images for each vertex in C.

We use a technique similar to the fpt-algorithm for a special case (when $H-C$ is an independent set) of (H, C, K)-COLORING problem: taking an edge xy where both x and y are not determined yet (neither in C nor in $H - C$), then either x or y should be in C since $H - C$ is an independent set. This technique easily results in an $O^*(2^k)$ algorithm for the PARAMETERIZED 3-COLORING problem. Our algorithm do better than that, by first applying this technique to any vertex with at least three undetermined neighbours, then when no such vertices

[1] $O^*(f(k))$ refers to $O(f(k)n^\alpha k^\beta)$ where α and β are constants independent of both n and k.

exist, determining the possible partitions of remaining undetermined vertices by enumeration. Our careful analysis gives an $O^*(1.713^k)$ upperbound for our fpt-algorithm, which can be also used to solve the special case of the (H, C, K)-coloring problem.

2 Preliminaries

Let $G = (V, E)$ be a *simple graph*, i.e., there are neither multiple edges between any two vertices nor self-loops in G. A *p-coloring* $(C_1, \cdots, C_p)$ of a vertex subset S in graph G is a partition of vertices of S into p disjoint subset such that $\sum_{i=1}^{p} C_i \subseteq S$[2] where C_i is an independent set for all i. A p-coloring $(C_1, \cdots, C_p)$ of S is a *complete p-coloring* of S if $\sum_{i=1}^{p} C_i = S$. A graph G is *p-colorable* if there is a complete p-coloring of V. Let V_1 and V_2 be vertex subsets in graph G such that $V_1 \subseteq V_2$, (C_1, C_2) be a 2-coloring of V_1, and (C_1', C_2') be a 2-coloring of V_2. Then (C_1', C_2') is *compatible* to (C_1, C_2) if $C_1 \subseteq C_1'$ and $C_2 \subseteq C_2'$.

Let S_B and S_I be two disjoint vertex subsets in graph G. A partition $[S_B, S_I]$ is a *bipartite-independent partition* if the subgraph induced by S_B is 2-colorable and S_I is an independent set. If $V = S_B + S_I$, then a bipartite-independent partition $[S_B, S_I = V - S_B]$ is a *complete* bipartite-independent partition. A bipartite-independent partition $[S_B', S_I']$ *k-extends* another bipartite-independent partition $[S_B, S_I]$ if $S_B \subseteq S_B'$, $S_I \subseteq S_I'$, and $|S_B'| \leq |S_B| + k$.

Let $P = x_1 \cdots x_p$ be a simple path induced by vertices $x_1, \cdots, x_p$, i.e., $x_i \neq x_j$ for $1 \leq i < j \leq p$. The *candidate set* of P is $\{x_2, x_4, \cdots, x_p\}$ if p is even, or $\{x_2, x_4, \cdots, x_{p-1}\}$ if p is odd. Let $C = x_1 \cdots x_p x_1$ be a simple cycle induced by vertices $x_1, \cdots, x_p$, i.e., $x_i \neq x_j$ for $1 \leq i < j \leq p$. The *candidate set* of cycle C is $\{x_2, x_4, \cdots, x_p\}$ if p is even, or $\{x_2, x_4, \cdots, x_{p-1}\} \cup \{x_p\}$ if p is odd.

3 Parameterized Algorithm

Let $[S_B, S_I]$ be a bipartite-independent partition of graph G. First we show two properties for the case when the subgraph induced by $V - S_B - S_I$ is a union of disjoint paths and cycles. We assume that any vertex $x \in V - S_B - S_I$ has no neighbours in S_I. This assumption will be justified later by our fpt-algorithm.

3.1 Two Properties When $G[V - S_B - S_I]$ Is a Union of Disjoint Paths/Cycles

Let $CS_1, \cdots, CS_q$ be the candidate sets of those paths/cycles in $G[V - S_B - S_I]$. When $G[V - S_B - S_I]$ is a union of paths/cycles, the first property is that at least $\sum_{i=1}^{q} |CS_i|$ vertices from the disjoint paths/cycles should be put into any complete bipartite-independent partition which k-extends $[S_B, S_I]$.

Lemma 1. *Let $[S_B', V - S_B']$ be a complete bipartite-independent partition. If $[S_B', V - S_B']$ k-extends $[S_B, S_I]$, then $V - S_B'$ contains at least $|CS_i|$ vertices from the path/cycle corresponding to candidate set CS_i, i.e., $k \geq \sum_{i=1}^{q} |CS_i|$.*

[2] In this paper, $+/-$ are used on sets to denote $\cup/\backslash$ for simplicity.

Proof. We number the paths/cycle corresponding to the candidate set CS_i by the index i of the candidate set. We first prove that for each path/cycle i, S'_B must contain at least $|CS_i|$ vertices from path/cycle i. There are two cases: CS_i is from either a path or a cycle.

If CS_i is from a path i, let path i be $x_1x_2\cdots x_p$ which is induced by vertices $x_1, x_2, \cdots, x_p$. By definition of candidate sets, $|CS_i| = |\{x_2, x_4, \cdots, x_p\}| = \lfloor \frac{p}{2} \rfloor$ if p is even, or $|CS_i| = |\{x_2, x_4, \cdots, x_{p-1}\}| = \lfloor \frac{p}{2} \rfloor$ if p is odd. For both cases, $|CS_i| = \lfloor \frac{p}{2} \rfloor$. Moreover, there are at least $\lfloor \frac{p}{2} \rfloor$ disjoint edges in path i: $x_1x_2, x_3x_4, \cdots$. Then each of those disjoint edges can have at most one vertex in $V - S'_B$, since $[S'_B, V - S'_B]$ is a complete bipartite-independent partition and $V - S'_B$ must be an independent set by the definition of bipartite-independent partition. Therefore, S'_B contains at least one vertex from each of those $\lfloor \frac{p}{2} \rfloor$ disjoint edges. It follows that S'_B contains at least $\lfloor \frac{p}{2} \rfloor = |CS_i|$ vertices from path i.

If CS_i is a from a cycle i, let cycle i be $x_1x_2\cdots x_px_1$ which induced by vertices $x_1, x_2, \cdots, x_p$. When p is even, we have that $|CS_i| = |\{x_2, x_4, \cdots, x_p\}| = \frac{p}{2}$. In the cycle i, there are $\frac{p}{2}$ disjoint edges: $x_1x_2, x_3x_4, \cdots, x_{p-1}x_p$. By the same arguments above for path i, S'_B contains at least one vertex from each of those disjoint edges, and thus contains at least $\frac{p}{2} = |CS_i|$ vertices from cycle i. Next we show that when p is odd, S'_B also contain at least $|CS_i|$ vertices from cycle i.

When p is odd, by definition we have that $|CS_i| = |\{x_2, x_4, \cdots, x_{p-1}\} \cup \{x_p\}| = \frac{p+1}{2}$ for cycle i. In this cycle of p vertices (p is odd), there are $\frac{p-1}{2}$ disjoint edges: $x_1x_2, \cdots, x_{p-2}x_{p-1}$. By the same arguments for path i, S'_B contains at least one vertex from each of those $\frac{p-1}{2}$ disjoint edges. If S'_B contains both vertices of one of those $\frac{p-1}{2}$ disjoint edges, S'_B contains at least $\frac{p-1}{2} + 1 = |CS_i|$ vertices from cycle i. Otherwise, S'_B contains exactly one vertex from each of those $\frac{p-1}{2}$ disjoint edges, and then $V - S'_B$ contains exactly one vertex from each of those disjoint edges. We have two cases:

Case 1: S'_B contains x_1. Since both S'_B and $V - S'_B$ contains exactly one vertex from each of those $\frac{p-1}{2}$ disjoint edges, $V - S'_B$ must contain x_2. It follows that S'_B must contain x_3 since $V - S'_B$ is an independent set, and then $V - S'_B$ must contain x_4. Repeat this, we will have that S'_B contains $x_1, x_3, \cdots, x_{p-2}$ and $V - S'_B$ contains $x_2, x_4, \cdots, x_{p-1}$. Since (1) x_{p-1} is in $V - S'_B$, (2) $V - S'_B$ is an independent set, and (3) there is an edge $x_{p-1}x_p$ in the cycle, S'_B must contain x_p. Therefore, S'_B contains $\frac{p-1}{2} + 1 = \frac{p+1}{2} = |CS_i|$ vertices: $\{x_1, x_3, \cdots, x_{p-2}\} \cup \{x_p\}$.

Case 2: S'_B does not contain x_1. Then $V - S'_B$ contains x_1. Since $V - S'_B$ is an independent set and there is an edge x_1x_p in cycle i, S'_B must contain x_p. Besides x_p, S'_B contains $\frac{p-1}{2}$ vertices from those $\frac{p-1}{2}$ disjoint edges $(x_1x_2, \cdots, x_{p-2}x_{p-1})$, by our assumption that S'_B contains exactly one vertex from each of those disjoint edge. Therefore, S'_B contains at $\frac{p-1}{2} + 1 = \frac{p+1}{2} = |CS_i|$ vertices from cycle i.

We have shown that S'_B contains at least $|CS_i|$ vertices from path/cycle i. It follows that S'_B contains at least $\sum_{i=1}^{q} |CS_i|$ vertices from $V - S_B - S_I$, since $G[V - S_B - S_I]$ is a union of disjoint paths/cycles. Note that S_B and $V - S_B - S_I$ are disjoint and CS_i are from $V - S_B - S_I$. It follows that $|S'_B| \geq |S_B| + \sum_{i=1}^{q} |CS_i|$. Moreover, because $[S'_B, V - S'_B]$ k-extends $[S_B, S_I]$, we have that $|S'_B| \leq |S_B| + k$. Therefore, $|S_B| + k \geq |S'_B| \geq |S_B| + \sum_{i=1}^{q} |CS_i|$, i.e., $k \geq \sum_{i=1}^{q} |CS_i|$. This completes our proof. □

The second property is that when a complete 2-coloring (C_1, C_2) of S_B is given, we can determine in polynomial time whether there is a complete bipartite-independent partition $[S'_B, V - S'_B]$ which k-extends $[S_B, S_I]$ and S'_B has a complete 2-coloring compatible to (C_1, C_2). If such a complete bipartite-independent partition exists, we construct a complete 2-coloring $(C'_1, C'_2)$of S'_B compatible to (C_1, C_2).

Lemma 2. *Let (C_1, C_2) be a complete 2-coloring of S_B. We can find a complete bipartite-independent partition $[S'_B, V - S'_B]$ k-extending $[S_B, S_I]$ such that S'_B has a complete 2-coloring compatible to C if there exists such one, or report 'NO' otherwise. This can be done in polynomial time.*

Proof. Figure 1 gives the algorithm to find the desired bipartite-independent partition if such one exists, or return 'NO' otherwise. It is obvious that the algorithm can terminate in polynomial time, since each step takes polynomial time.

```
**Algorithm-1**($G, k, [S_B, S_I], (C_1, C_2)$)
INPUT: a graph $G$, a parameter $k$, a bipartite-independent partition $[S_B, S_I]$, and a complete
2-coloring $(C_1, C_2)$ of $S_B$.
OUTPUT: either a complete bipartite-independent partition $[S'_B, V - S'_B]$ which $k$-extends
$[S_B, S_I]$ such that $S'_B$ has a complete 2-coloring compatible to $(C_1, C_2)$, or 'NO' otherwise.

1. if there is an edge $xy$ where $x, y \in V - S_B - S_I$ such that both $x$ and $y$ have a
neighbour in $C_1$ and another neighbour in $C_2$
        return 'NO';
2. foreach $x \in V - S_B - S_I$ which has neighbours in both $C_1$ and $C_2$, let $Y \subseteq V - S_B - S_I$
be neighbours of $x$, $Y_1 \subseteq Y$ be vertices which have neighbours in $C_1$, $Y_2 \subseteq Y$ be vertices
which have neighbours in $C_2$
        put $x$ into $S_I$ and $Y$ into $S_B$, $Y_1$ into $C_2$, $Y_2$ into $C_1$, and $k = k - |Y|$;
3. if $k < \sum_{path/cycle\ i} |CS_i|$
        return 'NO';
4. if there is a cycle $i$ of odd number vertices such that for each edge $xy$ in the cycle, both
$x$ and $y$ have neighbours in $C_1$ $(C_2)$
        returns 'NO';
5. foreach path/cycle $i$
        for a cycle of $p$ vertices where $p$ is odd, W.L.O.G, assume that $x_{p-1}x_p$ is an edge
where $x_{p-1}$ and $x_p$ have no neighbours in the same color set (either $C_1$ or $C_2$);
        put the candidate set $CS_i$ for the path/cycle $i$ into $S_B$ and other vertices $W_i$ of
path/cycle $i$ into $S_I$;
return $[S_B, S_I]$;
```

Fig. 1. Algorithm 1

When a vertex w has a neighbour in C_1 and neighbour in C_2, then w must be in $V - S'_B$, since $S_B + w$ has no 2-colorings compatible to (C_1, C_2). Then for Step 1, both x and y must be in $V - S'_B$, which is also impossible since $V - S'_B$ should be an independent set. Therefore, 'NO' is returned correctly at Step 1.

In Step 2, vertex x should be in $V - S'_B$ since x has neighbours in both C_1 and C_2. Then neighbours Y of x should be in S'_B for any complete bipartite-independent partition $[S'_B, V - S'_B]$ which k-extends $[S_B, S_I]$. Thus it is safe to put x into S_I and Y into S_B. Moreover, in any complete 2-coloring (C'_1, C'_2) of S'_B compatible to (C_1, C_2), $Y_1 \subseteq C'_2$ since vertices of Y_1 have neighbours in C_1. Similarly, $Y_2 \subseteq C'_1$. Finally, we need to reduce k to $k - |Y|$, since now we need to find a complete bipartite-independent partitions $(k - |Y|)$-extending $[S_B, S_I]$ after Step 2. This concludes that Step 2 is correct.

By our assumption that $G[V - S_B - S_I]$ is a union of paths/cycles. When $k < \sum_{path/cycle\ i} |CS_i|$, there are no complete bipartite-independent partitions k-extending $[S_B, S_I]$ by lemma 1. Therefore, Step 3 returns 'NO' correctly.

For a cycle i of p vertices where p is odd, $|CS_i| = \frac{p+1}{2}$. Let W be those vertices of cycle i which are also in S'_B where $[S'_B, V - S'_B]$ is a complete bipartite-independent partition k-extending $[S_B, S_I]$. Then by lemma 1, $|W| \geq |CS_i| = \frac{p+1}{2}$. This implies that at least two vertices x and y of W should be an edge xy of cycle i. However, when the condition of Step 4 is true, both x and y have neighbours in the same color set: either in C_1 or C_2. It contradicts that $W + S_B \subseteq S'_B$ is 2-colorable. Therefore, when the condition of Step 4 is true, no complete bipartite-independent partitions k-extending $[S_B, S_I]$ exist, and thus 'NO' is returned correctly.

To simplify discussions on the correctness of Step 5, let $[S'_B, S'_I]$ denote the $[S_B, S_I]$ returned at Step 5, and $[S_B, S_I]$ refers to the partition after Step 4.. Then $S'_B = S_B + \sum_{path/cycle\ i} CS_i$ and $S'_I = S_I + \sum_{path/cycle\ i} W_i = V - S'_B$.

By definition of CS_i, W_i in Step 5 is an independent set. Recall our assumption on $V - S_B - S_I$ as input: any vertex in $V - S_B - S_I$ has no neighbours in S_I (the assumption is made right before this subsection). Note that this assumption is still valid before Step 5. So $S'_I = S_I + \sum_{path/cycle\ i} W_i$ is still an independent set, since $W_i \subseteq V - S_B - S_I$. Moreover, $k >= \sum_{path/cycle\ i} |CS_i|$ after Step 3. It follows that $|S'_B| = |S_B| + |\sum_{path/cycl\ i} |CS_i| \leq |S_B| + k$. Therefore, we can conclude that $[S'_B, S'_I = V - S'_B]$ is a complete bipartite-independent partition k-extending $[S_B, S_I]$ such that S'_B has a complete 2-coloring compatible to (C_1, C_2) and thus Step 5 is correct, once we show that S'_B has a complete 2-coloring (C'_1, C'_2) compatible to (C_1, C_2).

To prove that, we first prove the following claim.

Claim: any vertex $y \in S_B$ such that y is not in $C_1 + C_2$, y has at most one neighbour in $V - S_I$.

Note that before Step 2 all vertices of S_B are in $C_1 + C_2$, since (C_1, C_2) is a complete 2-coloring of S_B before Step 2. So y must be put into S_B during Step 2, which implies that a neighbour $x \in V - S_B - S_I$ of y is put into S_I during Step 2. Recall that $G[V - S_B - S_I]$ is a union of disjoint paths/cycles before Step 2. It follows that y can have at most two neighbours in $V - S_B - S_I$ before Step 2. Moreover, y has no neighbours in S_B before Step 2. Otherwise, y should be in either C_1 or C_2 after Step 2. Since (1) y has no neighbours in S_B before Step 2, (2) y have at most two neighbours in $V - S_B - S_I$ before Step 2, and (3) one neighbour $x \in V - S_B - S_I$ of y is put into S_I during Step 2, it follows that

y has at most one neighbour in $V - S_I$ after Step 2, which conclude the proof of the Claim. □

Now we continue our proof of that S'_B has a complete 2-coloring (C'_1, C'_2) compatible to (C_1, C_2). Note that $S'_B = S_B + \sum_{path/cycle\ i} CS_i$ according to Step 5. Moreover, S_B may contains vertices other than those in $(C_1 + C_2)$ after Step 2. Let Y_i be those vertices in $S_B - (C_1 + C_2)$ which have neighbours in CS_i, and Z be those vertices in $S_B - (C_1 + C_2) - \sum_{path/cycle\ i} Y_i$ in Step 5. It is obvious that any vertex $x \in \sum_{path/cycle\ i} Y_i + Z$ has no neighbours in $C_1 + C_2$. Otherwise, x can be put into $C_1 + C_2$. By our definitions, it is also obvious that $S_B = (C_1 + C_2) + \sum_{path/cycle\ i} Y_i + Z$, and thus $S'_B = (C_1 + C_2) + \sum_{path/cycle\ i} Y_i + \sum_{path/cycle\ i} CS_i + Z$.

To show that $S'_B = (C_1 + C_2) + \sum_{path/cycle\ i} Y_i + \sum_{path/cycle\ i} CS_i + Z$ has a complete 2-coloring compatible to (C_1, C_2), we first show that $(C_1 + C_2) + \sum_{path/cycle\ i} CS_i$ has a 2-coloring (C^1_1, C^1_2) compatible to (C_1, C_2). Initially $C^1_1 = C_1$ and $C^1_2 = C_2$. Note that after Step 2, any vertex $x \in V - S_B - S_I$ has neighbours in at most one of C_1 and C_2. Otherwise, it should be processed in Step 2. Let x be a vertex of CS_i, then x has neighbours in at most one of $C^1_1 = C_1$ or $C^1_2 = C_2$, since CS_i is a subset of $V - S_B - S_I$. We put x into C^1_1 if it has a neighbour in C_2, or into C^1_2 otherwise. Note that $\sum_{path/cycle\ i} CS_i$ is an independent set, and then it is safe to put $\sum_{path/cycle\ i} CS_i$ into $C^1_2 + C^1_2$ as described above. Therefore, these operations find a 2-coloring (C^1_1, C^1_2) of $(C_1 + C_2) + \sum_{path/cycle\ i} CS_i$ which is compatible to (C_1, C_2).

Next We show that $(C_1 + C_2) + \sum_{path/cycle\ i} CS_i + \sum_{path/cycle\ i} Y_i$ has a complete 2-coloring (C^2_1, C^2_2) compatible to (C^1_1, C^1_2) of vertices $C^1_1 + C^1_2 = (C_1 + C_2) + \sum_{path/cycle\ i} CS_i$. Initially $C^2_1 = C^1_1$ and $C^2_2 = C^1_2$. Recall that alll vertices in $Y_i \subseteq S_B$ have no neighbours in $C_1 + C_2$. Then by our Claim, each vertex $y \in Y_i$ has exactly one neighbour in CS_i, and then no neighbours in $C_1 + C_2$ before Step 5. Since each vertex of CS_i is in C^1_1 or C^1_2 by our processing above, vertex $y \in Y_i$ can be put into $C^2_1(C^2_2)$ if its unique neighbour in CS_i is in $C^1_2(C^1_1)$. Therefore, all vertices in $(C_1 + C_2) + \sum_{path/cycle\ i} CS_i + \sum_{path/cycle\ i} Y_i$ has a complete 2-coloring which is compatible to the (C^1_1, C^1_2).

Finally, we show that $S'_B = (C_1 + C_2) + \sum_{path/cycle\ i} Y_i + \sum_{path/cycle\ i} CS_i + Z$ has a complete 2-coloring (C^3_1, C^3_2) compatible to (C^2_1, C^2_2) of those vertices in $(C_1 + C_2) + \sum_{path/cycle\ i} CS_i + \sum_{path/cycle\ i} Y_i = S'_B - Z$. Initially $C^3_1 = C^2_1$ and $C^3_1 = C^2_2$. Recall that vertices of Z have no neighbours in $C_1 + C_2 + \sum_{path/cycle\ i} CS_i$ by definition, and have no neighbours in $\sum_{path/cycle\ i} Y_i$ since by definition Y_i has only neighbours in CS_i. It follows that Z has no neighbours in $S'_B - Z$. That is, any vertex $z \in Z$ can have neighbours only in Z or $V - S'_B = S'_I$, i.e. have neighbours only in $Z + S'_I$. Recall again that Z is not in $C_1 + C_2$ by definition. Then by our Claim, any vertex $z \in Z$ has at most one neighbour in $V - S_I$, thus has at most one neighbour in Z and no neighbours in $(C_1 + C_2) + \sum_{path/cycle\ i} Y_i + \sum_{path/cycle\ i} CS_i$, since z can have neighbours only in $Z + S'_I$. This implies that the graph induced by Z is a set of disconnected edges and isolated vertices, thus Z is 2-colorable. Let (C'_1, C'_2) be a complete 2-coloring of Z. Then $(C^3_1 = C^2_1 + C'_1, C^3_2 = C^2_1 + C'_2)$ is a complete 2-coloring (C'_1, C'_2) of S'_B which is compatible to (C^2_1, C^2_2). By transitivity, (C^3_1, C^3_2) is compatible to (C_1, C_2). This completes our proof that Step 5 is correct, and then concludes our proof of this lemma. □

3.2 Main Algorithm

The main algorithm is presented in Figure 2. Next we prove that the main algorithm is correct.

```
Algorithm Param-3-Coloring($G, k, [S_B, S_I]$)
INPUT: a graph $G$, a parameter $k$, and a bipartite-independent partition $[S_B, S_I]$.
OUTPUT: either a complete bipartite-independent partition $[S'_B, V - S'_B]$ which $k$-extends
$[S_B, S_I]$, or 'NO' otherwise.

1. if $k < 0$ return 'NO';
   if $[S_B, V - S_B]$ is a bipartite-independent partition
        return $[S_B, V - S_B]$;
   if $k = 0$ return 'NO';
2. if $x \in V - S_B - S_I$ has a neighbour in $S_I$
       if $S_B + x$ is 2-colorable return Param-3-coloring($G, k - 1, S_B + x, S_I$);
       return 'NO';
3. if $x \in V - S_B - S_I$ in $S_B$ has two neighbours $y, z$ in $S_B$ where $yz$ is an edge in $G$, or
has no neigbhours in $V - S_B - S_I$
       return Param-3-coloring($G, k, S_B, S_I + x$);
4. if there is a vertex $x \in V - S_B - S_I$ which has three neighbours in $V - S_B - S_I$
       Let $N(x)$ be the neighbours of $x$ in $V - S_B - S_I$;
4.1    if $S_B + x$ is 2-colorable
           $[S'_B, V - S'_B]$ =Param-3-coloring($G, k - 1, S_B + x, S_I$);
           if $[S'_B, V - S'_B]! =$'NO'
               return $[S'_B, V - S'_B]$;
4.2    if $S_B + N(x)$ is 2-colorable
           return Param-3-coloring($G, k - |N(x)|, S_B + N(x), S_I + x$);
4.3    return 'NO';
   Let $CS_1, \cdots, CS_p$ be the candidate sets of paths/cycles induced by vertices in $V - S_B -$
$S_I$;
5. if $\sum_{i=1}^{p} |CS_i| > k$
       return 'NO';
6. if $k \leq |S_B|$
6.1    foreach enumeration of $W_1 \subseteq CS_1, \cdots, W_p \subseteq CS_p$ such that $[S_B + \sum_{i=1}^{p} W_i, S_I +$
$\sum_{i=1}^{p}(CS_i - W_i)]$ is a bipartite-independent partition
           Let $T \subseteq (V - S_B - S_I - \sum_{i=1}^{p} CS_i)$ be vertices having neighbours in $S_I +$
$\sum_{i=1}^{p}(CS_i - W_i)$
           if $[S'_B = S_B + \sum_{i=1}^{p} W_i + T, V - S'_B]$ is a complete bipartite-independent partition
and $|S'_B| \leq |S_B| + k$
               return $[S'_B, V - S'_B]$;
6.2    return 'NO';
7. foreach 2-coloring $(C_1, C_2)$ of $S_B$
       if there is a complete bipartite-independent partition $[S'_B, V - S'_B]$ which $k$-extends
$[S_B, S_I]$ such that there is a 2-coloring of $S'_B$ compatible to $(C_1, C_2)$
           return $[S'_B, V - S'_B]$;
8. return 'NO';
```

Fig. 2. The main algorithm

Lemma 3. *Algorithm Param-3-Coloring($G, k, [S_B, S_I]$) either finds a complete bipartite-independent partition $[S'_B, G - S'_B]$) which k-extends $[S_B, S_I]$ if such a bipartite-independent partition exists, or reports 'NO' otherwise.*

Proof. Without loss of generality, we can assume that the subgraph induced by S_B is a union of edges and isolated vertices.

Step 1 deals with the cases when the solution can be easily determined. First, if $k < 0$, there are no bipartite-independent partitions which k-extend $[S_I, S_I]$. Thus 'NO' is returned correctly. After this, we have that $k \geq 0$. If $[S_B, V - S_B]$ is a bipartite-independent partition, then $[S_B, V - S_B]$ is a complete bipartite-independent partition k-extending $[S_B, S_I]$ since $k \geq 0$, and thus $[S_B, V - S_B]$ is returned correctly. After this, $[S_B, V - S_B]$ is not a bipartite-independent partition. Therefore, any complete bipartite-independent partition extending $[S_B, S_I]$ must i-extend $[S_B, S_I]$ for some $i > 0$. It follows that 'NO' is returned correctly when $k = 0$ and $[S_B, V - S_B]$ is not a bipartite-independent partition. In conclusion, step 1 is correct.

After step 1, $k > 0$, $V - S_B - S_I$ is not empty, and S'_B must contain at least a vertex from $V - S_B - S_I$ for any complete bipartite-independent partition k-extending $[S_B, S_I]$. If a vertex $x \in V - S_B - S_I$ has a neighbour in S_I, then for any complete bipartite-independent partition $[S'_B, V - S'_B]$ k-extending $[S_B, S_I]$, x must be in S'_B and not in $V - S'_B$, since $V - S'_B$ should be an independent set and $S_I \subseteq V - S'_B$. If $S_B + x$ is 2-colorable, $[S_B + x, S_I]$ is a bipartite-independent partition, and then we only need to look for a complete bipartite-independent set $(k-1)$-extending $[S_B + x, S_I]$. Otherwise, 'NO' should be returned. Hence step 2 is correct.

Now we show that the first case of Step 3 is correct. In this case, vertex $x \in V - S_B - S_I$ has two neighbours y, z in S_B such that yz is an edge, then for any complete bipartite-independent partition $[S'_B, V - S'_B]$ k-extending $[S_B, S_I]$, x must be in $V - S'_B$, since S'_B should be 2-colorable. Thus we only need to search for a complete bipartite-independent set k-extending $[S_B, S_I + x]$. So this case is correct. Next we show that the second case of Step 3 is correct by proving that there is a complete bipartite-independent partition k-extending $[S_B, S_I]$ if and only if there is one k-extending $[S_B, S_I + x]$.

Let $[S'_B, V - S'_B]$ be a complete bipartite-independent partition k-extending $[S_B, S_I]$ where $x \in S'_B$. By definition, $V - S'_B$ is S_I plus an independent set $W \subseteq V - S_B - S_I$. Since vertex x has no neighbours in $V - S_B - S_I$ in this case, and after Step 2 x also has no neighbours in S_I, $S_I + W + x$, i.e., $V - S'_B + x$ is still an independent set. Therefore, $[S'_B - x, V - S'_B + x]$ is a complete bipartite-independent partition which k-extends $[S_B, S_I]$, since $[S'_B, V - S'_B]$ is a complete bipartite-independent partition k-extending $[S_B, S_I]$ and $x \notin S_B$. On the other hand, it is easy to see that any complete bipartite-independent partition k-extending $[S_B, S_I + x]$ also k-extends $[S_B, S_I]$ (note that $S_I + x$ is still an independent set). This conclude that the second of case of Step 3 is correct.

For any complete bipartite-independent partition $[S'_B, V - S'_B]$ k-extending $[S_B, S_I]$, vertex x in Step 4 is in either S'_B or $V - S'_B$. If x is in S'_B for a partition $[S'_B, V - S'_B]$, then $S_B + x \subseteq S'_B$ is 2-colorable, and step 4.1 should correctly find one which $(k-1)$-extends $[S_B + x, S_I]$. If Step 4.1 does not return any complete bipartite-independent partition, then for any complete bipartite-independent partition $[S'_B, V - S'_B]$, x should be in $V - S'_B$, and thus neighbours $N(x)$ of x should be in S'_B. This is possible only when $S_I + x$ is an independent set and $S_B + N(x)$ is 2-colorable. So when $S_B + N(x)$ is not 2-colorable, 'NO' is returned correctly at Step 4.3. Note that $S_I + x$ is indeed an independent set, since x has no neighbours in S_I after Step 2. Therefore, when $S_B + N(x)$ is 2-colorable, Step 4.2 correctly returns a complete bipartite-independent partition $[S'_B, V - S'_B]$ which $(k - |N(x)|)$-extends $[S_B + N(x), S_I + x]$ and also k-extends $[S_B, S_I]$ if there exists one. If Step 4.2 returns 'NO' when it does not find any target partition, it is still correct, since by our arguments above, there are no complete bipartite-independent partitions k-extending $[S_B, S_I]$ when both 4.1 and 4.2 can not find one.

After Step 4, any vertex $x \in V - S_B - S_I$ has at most two neighbours in $V - S_B - S_I$ because of Step 4, no neighbours in S_I because of Step 2, and no two neighbours in S_B which are neighbours of each other because of Step 3. Moreover, $G[V - S_B - S_I]$ now is a union of disjoint paths/cycles. According to lemma 1, for any complete bipartite-independent partition $[S'_B, V - S'_B]$ k-extending $[S_B, S_I]$, we have that $k \geq \sum_{i=1}^{p} |CS_i|$. Step 5 returns 'NO' correctly when $k < |\sum_{i=1}^{p} |CS_i|$.

Step 6.1 returns a correct complete bipartite-independent partition $[S'_B, V - S'_B]$ k-extending $[S_B, S_I]$ if it returns one, since it is a complete bipartite-independent partition. Next we show Step 6.2 returns 'NO' correctly by showing that if there is a complete bipartite-independent partition $[S'_B, V - S'_B]$ k-extending $[S_B, S_I]$, Step 6.1 can find a complete bipartite-independent partition $[S'_B, V - S'_B]$ for some $W_i \subseteq CS_i$ where $1 \leq i \leq p$.

Let $W_i = S'_B \cap CS_i$ for $1 \leq i \leq p$. Then $CS_i - W_i$ is in $V - S'_B$. By our notation, T are those vertices having neighbours in $S_I + \sum_{i=1}^{p}(CS_i - W_i)$. So T must be in S'_B. Let $Z = S'_B - S_B - \sum_{i=1}^{p} W_i - T$. We complete our proof that Step 6.1 can find a complete bipartite-independent partition k-extending $[S_B, S_I]$, once we show that $[S'_B - Z, V - S'_B + Z]$ is a complete bipartite-partition k-extending $[S_B, S_I]$. To show that, we first prove 2 claims.

Claim 1. Z is an independent set
Let R_i be the remaining vertices of path/cycle i which are not in $CS_i + T$, i.e., $\sum_{i=1}^{p} R_i = V - S_B - S_I - \sum_{i=1}^{p} CS_i - T$. By definition of CS_i, R_i is an independent set. Moreover, vertices in R_i have no neighbours in $R_j (i \neq j)$ since R_i and R_j are in two disconnected paths/cycles. So $\sum_{i=1}^{p} R_i$ is an independent set. Note that $Z \subseteq V - S_B - S_I - \sum_{i=1}^{p} CS_i - T = \sum_{i=1}^{p} R_i$. It follows that Z is an independent set. □

Claim 2. $V - S'_B + Z$ is an independent set
Note that $V - S'_B$ consists of three disjoint parts: (1) S_I, (2) $\sum_{i=1}^{p}(CS_i - W_i)$, and (3) vertices from $\sum_{i=1}^{p} R_i - Z$. First, $S_I + Z$ is an independent set, since both S_I and Z are independent sets and Z is subset of $V - S_B - S_I$ whose vertices have no neighbours in S_I after Step 2. Second, by definition of CS_i, W_i, T and R_i, $\sum_{i=1}^{p}(CS_i - W_i + \sum_{i=1}^{p} R_i)$ is also an independent set, and thus

$\sum_{i=1}^{p}(CS_i - W_i) + Z$ is an independent set since Z, as shown in the proof of Claim 1, is a subset of $\sum_{i=1}^{p} R_i$. Finally, $\sum_{i=1}^{p} R_i - Z + Z$ is an independent set by our argument above that $\sum_{i=1}^{p} R_i$ is an independent set. We conclude that $V - S'_B + Z$ is an independent set. □

Since $[S'_B, V - S'_B]$ is a complete bipartite-independent partition k-extending $[S_B, S_I]$, it is easy to see that $|S'_B - Z| \leq |S'_B| \leq |S_B| + k$ and $S'_B - Z$ is 2-colorable. Moreover, $V - S'_B + Z$ is an independent set by Claim 2. This concludes that $[S'_B - Z, V - S'_B + Z]$ is a complete bipartite-partition k-extending $[S_B, S_I]$, thus completing the proof of correctness of Step 6.

The partition $[S'_B, G - S'_B]$ returned at step 7 is correct, since it is a complete bipartite-independent partition k-extending $[S_B, S_I]$. Next we show that Step 8 returns 'NO' correctly.

If there is a complete bipartite-independent partition $[S'_B, G - S'_B]$ k-extending $[S_B, S_I]$, there is a 2-coloring (C'_1, C'_2) of S'_B. Let $C_1 = C'_1 \cap S_B$ and $C_2 = C'_2 \cap S_B$. By lemma 2, Step 7 should be able to find a complete bipartite-independent partition $[S''_B, G - S''_B]$ k-extending $[S_B, S_I]$ such that S''_B has a 2-coloring compatible to (C_1, C_2). Therefore, if Step 7 does not return a complete bipartite-independent partition $[S''_B, G - S''_B]$ k-extending $[S_B, S_I]$, there must be no complete bipartite-independent partitions k-extending $[S_B, S_I]$. This concludes that Step 8 returns 'NO' correctly. □

Next we analyze the complexity of algorithm Param-3-Coloring$(G, k, [S_B, S_I])$.

Lemma 4. *Assume $k \geq 0$. Algorithm Param-3-Coloring$(G, k, [S_B, S_I])$ terminates in time $O^*(1.466^k 1.365^{\frac{k+|S_B|}{2}})$.*

Proof. We use bounded-search tree analysis. Note that each step may directly return, or decrease the number of vertices in $G - S_B - S_I$, or decrease k. So the depth of the search tree is at most $n + k$. Moreover, the search along a particular root-leaf path can be done in polynomial time: Step 6 and 7 and be treated as inner nodes in the search tree, each enumeration in Step 6 can be done in polynomial time, and by lemma 2, each enumeration of Step 7 can be done in polynomial time. So the running time is dominated by the number of branches(leaves) in the search tree.

Next we prove that $3 \times 1.466^k 1.365^{\frac{k+|S_B|}{2}}$ is an upper bound on the number of branches. Only Step 4, 6 and 7 have branches. For Step 4, we have recursive equation:

$$f(k, |S_B|) = f(k-1, |S_B|+1) + f(k - |N(x)|, |S_B| + |N(x)|) \text{ where } |N(x)| \geq 3$$

For Step 6, we have recursive equation:

$$f(k \leq |S_B|, |S_B|) = 2^{\sum_{i=1}^{p} |CS_i|} \leq 2^k \text{ (because of Step 5)}$$

For Step 7, we have recursive equation:

$$f(k > |S_B|, |S_B|) = 2^{|S_B|}$$

When $k \leq 0$, the algorithm returns directly at step 1. So we have $f(k \leq 0, |S_B|) = 1$. We only need to show that the upper bound is correct for $k \geq 1$. The upper bound is correct for Step 4, since

$$f(1 \leq k < |N(x)|, |S_B|) \leq 3 \times 1.466^{k-1} 1.365^{\frac{k+|S_B|}{2}} + 1 \leq 3 \times 1.466^{k} 1.365^{\frac{k+|S_B|}{2}}.$$

$$\begin{aligned} f(k \geq |N(x)|, |S_B|) &\leq 3 \times 1.466^{k-1} 1.365^{\frac{k+|S_B|}{2}} + 3 \times 1.466^{k-|N(x)|} 1.365^{\frac{k+|S_B|}{2}} \\ &\leq 3 \times 1.466^{k} 1.365^{\frac{k+|S_B|}{2}} \text{ (note that } |N(x)| \geq 3). \end{aligned}$$

The upper bound is correct for Step 6 and 7, since

$$f(k \leq |S_B|, |S_B|) = 2^{\sum_{i=1}^{p} |CS_i|} \leq 2^k \leq 3 \times 1.466^k 1.365^{\frac{k+|S_B|}{2}} (k \leq |S_B|), \text{ and}$$

$$f(k > |S_B|, |S_B|) = 2^{|S_B|} \leq 3 \times 1.466^k 1.365^{\frac{k+|S_B|}{2}} (k > |S_B|)$$

Since there are at most $3 \times 1.466^k 1.365^{\frac{k+|S_B|}{2}}$ branches, each of which can be done in $O(n^c)$ time where c is a constant independent of k, algorithm Param-3-Coloring$(G, k, [S_B, S_I])$ terminates in time $O^*(1.466^k 1.365^{\frac{k+|S_B|}{2}})$.

Now we are ready to apply our algorithm to solve the PARAMETERIZED 3-COLORING problem.

Theorem 1. *Param-3-Coloring($G, k, [\phi, \phi]$) solves the* PARAMETERIZED 3-COLORING *problem correctly in time $O(1.713^k n^{O(1)})$.*

Proof. By lemma 3, Param-3-Coloring$(G, k, [\phi, \phi])$ either finds a complete bipartite-independent partition $[S'_B, G - S'_B]$ which k-extends $[\phi, \phi]$ if such a bipartite-independent partition exists, or reports 'NO' otherwise. If the algorithm finds one $[S'_B, G - S'_B]$, we have that S'_B contains at most k vertices and is 2-colorable, and that $G - S'_B$ contains the remaining vertices and is an independent set. So Param-3-Coloring$(G, k, [\phi, \phi])$ solves the PARAMETERIZED 3-COLORING problem correctly.

By lemma 4, Param-3-Coloring$(G, k, [\phi, \phi])$ terminates in time $O^*(1.466^k 1.365^{\frac{k}{2}}) = O^*(1.713^k)$. This concludes our proof.

4 Future Work

Given a graph G, if there exists a 3-coloring of G such that $k \leq 0.527n$, our algorithm is faster than the best algorithm to solve the 3-COLORING problem. It is interesting to improve our algorithm to beat the best algorithm for the 3-COLORING problem. There are some challenging questions. Can we solve the cases in Step 6 and 7 is polynomial time or better than 2^k and $2^{|S_B|}$? What kind graphs have a large independent set such that the remaining graph is 2-colorable? We have studied the PARAMETERIZED 3-COLORING problem on vertex coloring. How about edge coloring?

Another direction is to consider different parameters. One possible candidate is the size of an independent size. Can we find an independent set of size k such that the remaining graph is 2-colorable? If we do not require the k vertices to be an independent set, there is an FPT algorithm of time $O^*(3^k)$ [13]. When we require the k vertices to be an independent set, is the problem still fixed-parameter tractable?

References

1. Beigel, R., Eppstein, D.: 3-coloring in time $O(1.3446^n)$: a no-MIS algorithm. In: Proc. 36th Symp. Foundations of Computer Science, pp. 444–453 (1995)
2. Beigel, R., Eppstein, D.: 3-coloring in time $O(1.3289^n)$. Journal of Algorithms 54(2), 168–204 (2005)
3. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM J. on Computing 39(2), 546–563 (2009)
4. Blum, A., Karger, D.: An $O(n^{3/14})$-coloring algorithm for 3-colorable graphs. Information Processing Letter 61(1), 49–53 (1997)
5. Byskov, J.M.: Enumerating maximal independent sets with applications to graph colouring. Operations Research Letters 32, 547–556 (2004)
6. Chaitin, G.J.: Register allocation & spilling via graph coloring. In: Proc. of the 1982 SIGPLAN Symposium on Compiler Construction, pp. 98–105 (1982)
7. Díaz, J., Serna, M., Thilikos, D.M.: (H, C, K)-coloring: fast, easy, and hard cases. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 304–315. Springer, Heidelberg (2001)
8. Eppstein, D.: Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In: Proc. 12th Symp. on Discrete Algorithms, pp. 329–337 (2001)
9. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. J. Graph Algorithms and Applications 7(2), 131–140 (2003)
10. Khanna, S., Linal, N., Safra, S.: On the hardness of approximating the chromatic number. In: Proc. 2nd Isral Symp. on Theory and Computing Systems, pp. 256–260 (1993)
11. Lawer: A note on the complexity of the chromatic number problem. Information Processing Letter 5(3), 66–67 (1976)
12. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. In: Proc. 25th Symp. of Theory of Computing, pp. 286–293 (1993)
13. Reed, B.A., Smith, K., Vetta, A.: Finding odd cycle transversals. Operation Research Letters 32(4), 299–301 (2004)
14. Schiemeyer, I.: Deciding 3-colourability in less than $O(1.415^n)$ steps. In: Proc. 19th Int. Workshop Graph-Theoretic Concepts in Computer Science, pp. 177–182 (1994)

Kinetic Red-Blue Minimum Separating Circle

Yam Ki Cheung, Ovidiu Daescu[*], and Marko Zivanic

Department of Computer Science
The University of Texas at Dallas
Richardson, TX USA
{ykcheung,daescu,mxz052000}@utdallas.edu

Abstract. In this paper, we study a kinetic version of the red-blue minimum separating circle problem, in which some points move with constant speed along straight line trajectories. We want to find the locus of the minimum separating circle over a period of time. We first consider two degenerate cases of this problem. In the first one (P1), we study the minimum separating circle problem with only one mobile blue point, and in the second one (P2), we study the minimum separating circle problem with only one mobile red point. Then, we give a solution for the general case (P3), in which multiple points are mobile.

1 Introduction

Let $\mathcal{R}$ and $\mathcal{B}$ be two finite sets of points in R^2, of size $|\mathcal{R}| = n$ and $|\mathcal{B}| = m$, respectively. We refer to $\mathcal{R}$ as the set of red points and to $\mathcal{B}$ as the set of blue points. Given a family $\mathcal{F}$ of curves in R^d space, $\mathcal{R}$ and $\mathcal{B}$ are $\mathcal{F}$ separable if there exists a curve $f \in \mathcal{F}$ such that each connected component of the R^d space partitioned by f contains only red points or only blue points, but not both. The curve f is called a separator for sets $\mathcal{R}$ and $\mathcal{B}$. Different decision and optimization separability problems have been studied for a variety of separators such as lines, circles, convex polygons, strips, etc.

In [6] the authors define a constrained version of the circular separability problem, called the *minimum separating circle problem*, as follows. Let $\mathcal{S}$ denote the set of circles such that each circle in $\mathcal{S}$ encloses all points in $\mathcal{R}$ while having the smallest number of points of $\mathcal{B}$ in its interior. The goal is to find the smallest circle in $\mathcal{S}$, called the *minimum separating circle* and denoted by $C_{\mathcal{B}}(\mathcal{R})$. See Figure 1 for an illustration.

The problem addresses an issue that arises in military planning. It can be used to determine the best location to deploy an explosive and the amount needed such that all enemy forces, represented by red points, will be impacted while making sure that the number of civilian objects in the blast radius is minimized. It is also applicable in determining the best set-up of communication devices such that all red devices stay connected and as few blue devices as possible can intercept their communication. Two algorithms for the static version of this problem have been proposed in [6].

[*] This research was partially sponsored by NSF award CNS-1035460.

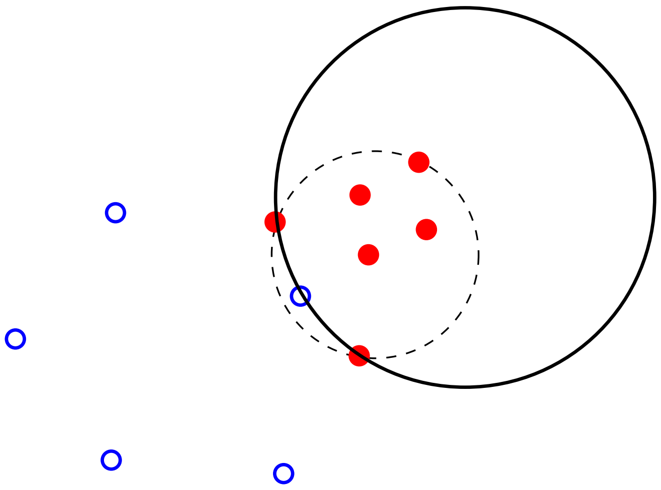


Fig. 1. Minimum red enclosing circle (dashed) and red-blue separating circle (solid)

In this paper, we study a kinetic version of the red-blue minimum separating circle problem, in which some points move with constant speed along straight line trajectories. We want to find the locus of the minimum separating circle over a period of time. We first consider two degenerate cases of this problem. In the first one (P1), we study the minimum separating circle problem with only one mobile blue point, and in the second one (P2), we study the minimum separating circle problem with only one mobile red point. Then, we give a solution for the general case (P3), in which multiple points are mobile.

For the case when two (static) point sets can be separated by a circle, Fisk [11] gave a quadratic time and space algorithm to compute the minimum separating circle. The result was later improved to optimal linear time and space by O'Rourke et. al. [16]. The linear separability problem, in which the separator is a hyperplane, reduces to linear programming, which in turn can be solved in linear time for any fixed dimension using Megiddo's algorithm [15]. Aronov et. al. [2] considered the linear separability problem for point sets that may be inseparable. When the points are linearly partitioned into two parts, each part may contain some misclassified points. The authors defined four different metrics for measuring the error of classification based on the misclassified points and gave solutions for finding the best separator which minimizes the error under each metric.

Other types of separators besides lines and circles have been considers recently. For example, it was shown that separability problems in R^2 using a double wedge [13], a wedge, or a strip [14] can be solved in $O(n \log n)$ time. Arkin et al. [8] proved an $\Omega(n \log n)$ lower bound for many of these two-dimensional separability problems. Fekete [10] showed that the problem of separating two planar point sets by a convex polygon with minimum number of edges is NP-complete. In [9], Hurtado et al. studied and gave cubic or slightly super cubic algorithms for various separability problems of two disjoint sets in R^3.

To the best of our knowledge the kinetic version of the minimum separating circle problem, as defined in this paper, has not been studied yet. However, there is a significant number of publications on related topics. Atallah [3] introduced the concept of kinetic computational geometry in a seminal paper on this topic. Basch et al. [5] introduced a set of kinetic data structures that can be used to maintain the convex hull of a moving set of points. Ross [18] gives an algorithm for maintaining the nearest-point Voronoi diagram of a kinetic data set. He presents an update algorithm for the topological structure of the Voronoi diagram of moving points, using $O(\log n)$ time for each change. Demain et al. [7] presented a kinetic data structure that finds the minimum spanning circle for a moving set of points. The efficiency of their data structure is $O(n^{1+\epsilon})$, which is the best that can be achieved for a data structure based on Delaunay triangulation. Rahmati et al. [17] presented a kinetic data structure for the maintenance of the minimum spanning tree on a set of moving points in 2-dimensional space.

Our result: We show that (1) the locus of the center of $C_{\mathcal{B}}(\mathcal{R})$ with one moving blue point (P1) has a complexity of $O(mn)$ and can be found in $O(mn\log(mn))$ time, (2) the locus of the center of $C_{\mathcal{B}}(\mathcal{R})$ with one moving red point (P2) has a complexity of $O(m^2n)$ and can be found in $O(m^2n\log m)$ time, and (3) the locus of the center of $C_{\mathcal{B}}(\mathcal{R})$ with multiple moving blue points and red points has a complexity of $O(m^2n^{2+\epsilon})$ and can be found in $O(m^2n^{2+\epsilon}\log(mn)+mn^{3+\epsilon})$ time.

2 Preliminaries

If not mentioned otherwise, we assume every point is either stationary or moving along a linear trajectory with constant speed and all stationary points are in general position.

Let $P=\{p_1,p_2,\ldots,p_n\}$ be a set of n points in $\mathcal{R}^2$. Let $B(p_i,p_j)=\{x\in\mathcal{R}^2 \mid |xp_i|=|xp_j|\}$ be the perpendicular bisector of p_i and p_j, where $|xp_i|$ is the distance between x and p_i. For the farthest point Voronoi diagram, the Voronoi cell $\mathcal{V}(p_i)$ for a point p_i is the set of points $q\in\mathcal{R}^2$ that are farther to p_i than to any other points in P:

$$\mathcal{V}(p_i)=\{q \mid |p_iq|>|p_jq|, \forall j\neq i\}.$$

It is known that only points which are vertices of the convex hull of P can have Voronoi cells. The vertices of the Voronoi cells are called Voronoi vertices and edges of the Voronoi cells, which are portions of bisectors of points in P, are called Voronoi edges. We denote by $FVD(P)=\{\mathcal{V}(p_i) \mid p_i\in P\}$ the farthest point Voronoi diagram of P. We treat $FVD(P)$ as a tree, with the center of the minimum enclosing circle of P as its root.

The farthest point Delaunay triangulation $DT(P)$ is a dual graph of the farthest point Voronoi diagram. Every vertex in $DT(P)$ corresponds to a cell in $FVD(P)$. Two vertices of $DT(P)$ are connected if the two corresponding Voronoi cells have a common edge. We summarize the properties of the farthest point Delaunay triangulation:

Property 1. Given a set P of points in general position, let $c(p_i, p_j, p_k)$ be the center of the circumcircle $C(p_i, p_j, p_k)$ of three points p_i, p_j, p_k in P. Then triangle $\triangle(p_i, p_j, p_k) \in DT(P) \Leftrightarrow c(p_i, p_j, p_k)$ is a vertex of $FVD(P) \Leftrightarrow C(p_i, p_j, p_k)$ encloses all points in P.

Given a system of multiple moving objects, kinetic data structures are a class of algorithmic techniques that efficiently maintain geometric structures such as the Voronoi diagram, minimum enclosing circle, convex hull, etc. Each geometric structure maintained has a combinatorial description that changes only at discrete times, when certain events occur. Hence, the combinatorial description of the geometric structure can be maintained by an event queue driven simulation of the motion of objects. The key idea on developing kinetic data structures is called *animating proofs through time* [12], which maintains a set of elementary conditions on the moving data called *certificates* to prove that the geometric structure is correctly constructed over time. Events which change the combinatorial description are referred to as *external* events. Events that are not external are referred to as *internal* events.

Next, we briefly discuss two algorithms proposed in [6] to solve the static version of the minimum separating circle problem. The first algorithm is based on a sweep procedure on edges of $FVD(\mathcal{R})$, while the second algorithm is based on circular range counting queries.

Given a set of static red points $\mathcal{R}$ and a set of static blue points $\mathcal{B}$, in [6], the authors showed that the minimum separating circle is either the minimum enclosing circle $MEC(\mathcal{R})$ of $\mathcal{R}$ or the circumcircle of two red points and one blue. It follows that the center of the minimum separating circle is either a Vertex of $FVD(\mathcal{R})$ or lies on an edge of $FVD(\mathcal{R})$.

Consider a Voronoi edge e_{ij}, defined by two red points r_i and r_j. The first algorithm in [6] starts on e_{ij} by constructing an enclosing circle C of $\mathcal{R}$ which passes through r_i and r_j and has the smallest possible radius. The center c of C is one endpoint of e_{ij}. Then, C is grown by sweeping c along e_{ij} and keeping r_i and r_j on the circumference of C.

A point $E \in e_{ij}$ is an event point if, when c sweeps through E, the circle C sweeps through a blue point (see Figure 2 for an illustration). The number of blue points enclosed by C is updated at each event point. The sweep procedure on e_{ij} terminates when the other endpoint of e_{ij} is reached.

Given that the sweep algorithm starts with the smallest possible enclosing circle for a given edge of $FVD(\mathcal{R})$ and strictly increases the radius of that circle looking for events where there are fewer points from $\mathcal{B}$ in the interior of the circle, it is only necessary to check when a blue point exits the circle. Thus, only the *exit* event points, where a blue point exits the enclosing circle need to be analyzed. It is shown in [6] that a blue point in $\mathcal{B}$ defines at most one exit event point on $FVD(\mathcal{R})$, and such exit event point can be computed in $O(\log n)$ time.

The second algorithm in [6] finds the minimum separating circle by examining exit event points only. That is, for each point $b \in \mathcal{B}$, the algorithm computes the exit event point associated with b. Then, a circular range counting query is performed for the *candidate separating circle*, which centers at the exit event

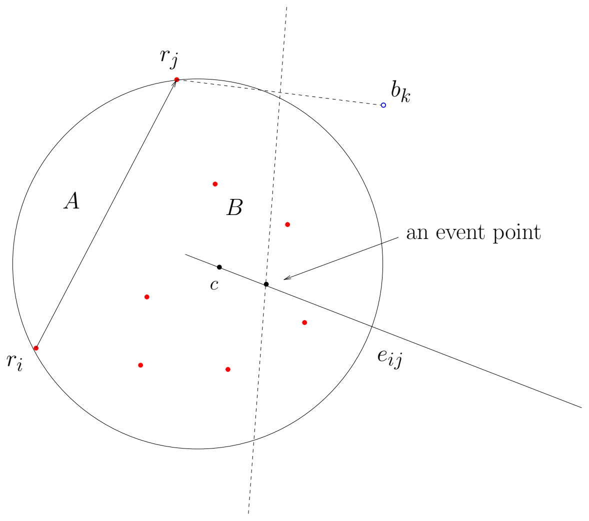


Fig. 2. Finding an event point on edge e_{ij}

point and passes through the corresponding blue point, to compute the number of blue points enclosed.

3 General Approach

In this section, we give an overview of our approach that applies for the kinetic minimum separating circle problem. At any time instant t, the minimum separating circle is either the minimum enclosing circle of the red points or one of the candidate separating circles, i.e. circles that center at an exit event point and pass through the corresponding blue point. Hence, given any time instant t, we can find the minimum separating circle by computing the $FVD(\mathcal{R})$ as well as $O(m)$ exit event points. Then, we count the number of blue points enclosed by each candidate separating circle. To find the locus of the minimum separating circle over a period of time, we need to keep track of the (location and size) of each candidate circle as well as the number of blue points enclosed by each candidate circle. These can be achieved by maintaining several kinetic data structures, such as the kinetic farthest point Voronoi diagram of $\mathcal{R}$ (and the kinetic farthest point Delaunay triangulation), the trajectory of each exit event point, and the number of blue points enclosed by each candidate circle. We first illustrate this approach for P1 and P2, then apply it for the general case, P3.

4 The Minimum Separating Circle with One Mobile Blue Point

In this section, we consider the case when only one blue point is moving with constant speed along a straight line. We formally define the problem as follows:

Let $\mathcal{R}$ be a set of n fixed red points, let S be a set of $m-1$ blue points, and let p be a mobile blue point moving with constant speed along a straight line. Let $\mathcal{B} = S \cup p$. We want to find the locus of the center of the minimum separating circle $C_{\mathcal{R}}(\mathcal{B})$ over a period of time.

We first study a structure called *exit region*, which was introduced in [6] and plays a key role in solving the kinetic version of the minimum separating problem. In the sweep algorithm introduced in [6], the center c of a separating circle C is swept along a Voronoi edge, while keeping the red points which define the Voronoi edge on it boundary. Certain region of the circle C at its initial state is excluded during the sweep. We define the exit region associated with a Voronoi edge of $FVD(R)$ as the union of points excluded from C during the execution of the sweep algorithm on this edge. See Fig. 3 for an illustration. More specifically, let a and b be the two end points of the Voronoi edge. Let C_a (resp. C_b) be the circle passing through the three red points, which define a (resp. b) with C_a smaller than C_b. Then the exit region associated with this Voronoi edge is $C_a \setminus C_b$. Only points within $MEC(\mathcal{R}) \setminus CH(\mathcal{R})$ can be excluded by a sweep, where $CH(\mathcal{R})$ is the convex hull of $\mathcal{R}$. All exit regions must lie within $MEC(R) \setminus CH(R)$. It is known that all exit regions are piecewise disjoint [6].

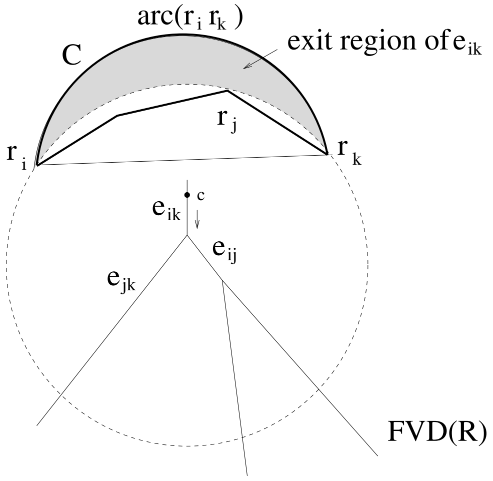


Fig. 3. The exit region associated with a Voronoi edge

Lemma 1. *Exit regions give a partition of $MEC(\mathcal{R})/CH(\mathcal{R})$, which is a dual of $FVD(\mathcal{R})$.*

Proof. Since all exit regions are piecewise disjoint, exit regions give us a partition of $MEC(R)/CH(R)$. Notice that there is a one-to-one correspondence between Voronoi edges and exit regions. Each exit region of a given edge is bounded by two circles. Each circle is the circumcircle of a Delauney triangle, whose center defines one end point of the Voronoi edge. Two Voronoi edges are incident to the same vertex if and only if their exit regions share a common arc. □

Corollary 1. *Given the farthest point Voronoi diagram of $\mathcal{R}$, we can compute all exit regions in linear time.*

Given a blue point $b \in \mathcal{B}$, if b is enclosed by an exit region associated with a Voronoi edge e_{ij}, which is a part of the perpendicular bisector between two red points r_i and r_j, b generates an exit event point on e_{ij} and the exit event point is the center of the circumcircle $C(b, r_i, r_j)$ of b, r_i, and r_j. If b moves with constant speed along a straight line, the exit event point moves along e_{ij}.

Let $b(t) = \overrightarrow{m}t + q$ be the trajectory of b, where $\overrightarrow{m}$ is a constant vector and q is a point. The perpendicular bisector of b and r_i is

$$B_{b,r_i}(t,u) = \frac{b(t) + r_i}{2} + \overrightarrow{m'(t)}u,$$

where $m'(t) = (y_b(t) - y_{r_i}, x_{r_i} - x_b(t))^T$ is a vector which defines the slope of the bisector, and $u \in R$. The center of the circumcircle $C(b, r_i, r_j)$ lies on the bisector between $b(t)$ and r_i at

$$u(t) = \frac{(r_j - r_i) \cdot (r_j - b(t))}{2\overrightarrow{m'(t)} \cdot (r_j - r_i)},$$

where " $\cdot$ " denotes the inner product between two vectors. Hence, the trajectory of the exit event point of b, i.e. the center of $C(b, r_i, r_j)$, is

$$\frac{b(t) + r_i}{2} + \overrightarrow{m'(t)}\frac{(r_j - r_i) \cdot (r_j - b(t))}{2\overrightarrow{m'(t)} \cdot (r_j - r_i)} \qquad (*)$$

Lemma 2. *If the trajectory of the mobile blue point is a straight line, the trajectory of the corresponding mobile exit event point is a cyclic path on $FVD(\mathcal{R})$.*

Proof. Obviously, the trajectory of an exit event point is a continuous curve. We prove this lemma by considering two cases. *Case one:* the trajectory of the mobile blue point p is a line intersecting both $CH(\mathcal{R})$ and $MEC(\mathcal{R})$. When p enters $MEC(\mathcal{R})$, it creates an exit event point at the root of $FVD(R)$, which moves along the edge defining the first exit region it visits. When p enters a new exit region by crossing a circumcircle of three red points, the exit event point crosses a vertex of $FVD(R)$ which is the center of the circumcircle that separates these two exit regions and, moves to a new Voronoi edge. Eventually, the exit event point visits an unbounded Voronoi edge, when it enters an exit region bounded by an edge of $CH(\mathcal{R})$ and one circumcircle, and moves to ∞ as it approaches the boundary of $CH(\mathcal{R})$. Note that p does not have an exit event point when $p \in CH(\mathcal{R})$. Similarly, when p exits from $CH(\mathcal{R})$, its exit event point re-appears at ∞, and travels along a path on $FVD(\mathcal{R})$, and eventually reaches the root of $FVD(R)$ when p crosses $MEC(\mathcal{R})$ the second time to form a closed path.

Case two: the trajectory of p is a line intersecting $MEC(\mathcal{R})$ but not $CH(\mathcal{R})$. Let r_i and r_j be two red points which define a Voronoi edge e_{ij} in $FVD(\mathcal{R})$. The exit region associated with e_{ij} is bounded by two circular arcs, which are both incident to r_i and r_j. It follows that if the trajectory of p intersects one arc of the exit region, it must intersect the arc exactly twice. Otherwise the trajectory intersects the line segment connecting r_i and r_j, which is enclosed by $CH(\mathcal{R})$. Note that r_i and r_j must be vertices of $CH(\mathcal{R})$. This implies that the trajectory of the corresponding exit event point starts at the root of $FVD(\mathcal{R})$ and traverses a path along edges of $FVD(\mathcal{R})$ and returns back to the root following the same path. □

Next, we give details of the solution for this problem. Observe that each fixed blue point defines at most one static exit event point, which in turn defines a fixed candidate separating circle. The number of blue points enclosed by such candidate separating circle changes only when the mobile blue point p enters or leaves the circle. The mobile blue point p defines a mobile exit event point. Not only the center and radius of the corresponding candidate circle changes continuously, the number of blue points enclosed by the corresponding candidate separating circle changes over time, as well. We need to dynamically maintain two classes of structures: (1) the trajectory of exit event points, and (2) the number of blue points enclosed by each candidate separating circle over time. We need to analyze the following events and update the corresponding structures accordingly.

case 1) the mobile blue point enters or leave a static candidate circle,
case 2) the exit event point associated with the mobile blue point moves to a new edge of the farthest neighbor Voronoi diagram $FVD(R)$ of $\mathcal{R}$, and
case 3) the candidate circle associated with the mobile blue point encloses or excludes a new blue point.

To avoid ambiguity, we refer to these events as instant events, distinguishing from the event points introduced in Section 2.

Note that case 1 instant events maintain the count of blue points enclosed by each fixed candidate circle. Case 2 instant events allow us to track the trajectory of the exit event point associated to the mobile blue point. Case 3 instant events define the moment when the number of blue points enclosed by the candidate separating circle associated with the mobile blue point changes.

Case 1 instant events: These are the instant events when the mobile blue point enters or leave a fixed candidate circle.

Lemma 3. *There are $O(m)$ case 1 instant events, and can be found in constant time each, given that all fixed exit event points are known.*

Proof. Trivial. Given a fixed candidate circle, the time at which a mobile point with know trajectory enters or leaves the circle can be determined in constant time. Also since the mobile blue point follows a linear trajectory, it may enter/leave a circle at most once. □

Case 2 instant events: These are the instant events when the exit event point associated to the mobile blue point moves to a new edge of the farthest neighbor Voronoi diagram $FVD(R)$ of R.

Lemma 4. *We have $O(n)$ case 2 instant event, which can be computed in $O(n)$ time.*

Proof. Obviously, if the mobile blue point enters an exit region, it creates an exit event point on the corresponding Voronoi edge. As a result, a case 2 instant event is created when the mobile blue point crosses the boundary between two exit regions. Since the mobile blue point follows a linear path, it can traverse at most $O(n)$ exit regions and enter the same exit region no more than twice. Hence, we have at most $O(n)$ case 2 instant events. The time instant when the mobile blue point enters or leaves an exit region can be determined in constant time given that the exit regions are pre-computed and all stationary red points are in general position. □

The trajectory of the mobile exit event point on a Voronoi edge can be computed in constant time using equation (*) once the corresponding case 2 instants events are computed.

Case 3 instant events: These are the instant events when the candidate circle associated to the mobile blue point encloses or excludes a blue point.

Lemma 5. *There are $O(mn)$ case 3 instant events.*

Proof. The candidate separating circle of the mobile blue point will enclose/enclude a fixed blue point when its center, i.e the mobile exit event point, sweeps across an event point of a fixed blue point along its path. Each Voronoi edge can have at most $O(m)$ event points [6], and the result follows. □

Thus the trajectory of each exit event point and the count of blue points enclosed by each candidate circle are maintained over time. We will give a procedure to compute the trajectory of the minimum separating circle in the next section.

Theorem 1. *The locus of the minimum separating circle of $\mathcal{R}$ and $\mathcal{B}$ has a complexity of $O(mn)$ can be computed in $O(mn\log(mn))$ time.*

5 The Minimum Separating Circle with One Mobile Red Point

In this section, we study the minimum separating circle problem for two point sets $\mathcal{R}$ and $\mathcal{B}$, such that all points are stationary except one red point p, which is moving along a linear trajectory with constant speed. We show how to find the locus of the center of the minimum separating circle $C_{\mathcal{R}}(\mathcal{B})$.

Let $b \in \mathcal{B}$, and assume b is enclosed by an exit region associated to a Voronoi edge e, which is defined by a fixed red point r and the mobile red point p, then the exit event point of b moves along the perpendicular bisector between b and r and its trajectory has a formula similar to (*). The difficulty to track the trajectory of each exit event point lies on the fact that the exit regions change dynamically, i.e. some exit regions will disappear and some new exit regions will be formed, and boundaries of some exit regions change continuously due to the mobile red point. Fortunately, we do not have to compute all exit regions formed over time. Since $FVD(\mathcal{R})$ is a dual graph of the union of the exit regions, we can maintain the topology of the farthest point Voronoi diagram $FVD(\mathcal{R})$ of $\mathcal{R}$ instead. Note that only points on the convex hull $CH(\mathcal{R})$ of $\mathcal{R}$ contribute to $FVD(\mathcal{R})$, and it is necessary to maintain the convex hull of $\mathcal{R}$. Knowing $FVD(\mathcal{R})$ allows us to compute the moments when some blue point is enclosed by a different exit region, which is a crucial step to maintain the trajectory of each event point. We also need to maintain the number of blue points enclosed by each candidate separating circle. In summary, we need to maintain the following four classes of kinetic data structures: the (topology of the) convex hull of $\mathcal{R}$, the (topology of the) farthest point Voronoi diagram of $\mathcal{R}$, the trajectory of each exit event point and the number of blue points enclosed by each candidate circle. To achieve this, we define the following four classes of instant events:

case 1) the appearance/disappearance of a vertex on the boundary of the convex hull $CH(\mathcal{R})$ of $\mathcal{R}$,
case 2) the appearance/disappearance of a vertex on the farthest neighbor Voronoi diagram $FVD(\mathcal{R})$ of $\mathcal{R}$,
case 3) the exit event point associated with a blue point moves to another Voronoi edge, and
case 4) the candidate separating circle associated with an exit event point encloses/excludes a blue point.

Case 1 instant events: These are time instants when a vertex of $CH(R)$ appears or disappears.

Lemma 6. *There are $O(n)$ instant events in case 2, and each event can be identified in constant time given the convex hull of fixed red points $CH(\mathcal{R} \setminus p)$ is known.*

Proof. Case 1 instant events occur when p passes through the intersection between a line supporting some edge of $CH(\mathcal{R} \setminus p)$ and the trajectory of p. Hence, each instant event can be identified in constant time. See Fig. 4 for an illustration. □

When a case 1 instant event is triggered, not only the convex hull of $\mathcal{R}$ but the topology of the farthest point Voronoi diagram of $\mathcal{R}$ changes as well.

Lemma 7. *The topology of the farthest point Voronoi diagram $FVD(\mathcal{R})$ of $\mathcal{R}$ can be updated in constant time for each case 1 instant event.*

Proof. Instead of working on $FVD(\mathcal{R})$ directly, we turn our attention to the dual graph of $FVD(\mathcal{R})$, the Delaunay triangulation $DT(\mathcal{R})$. When a vertex

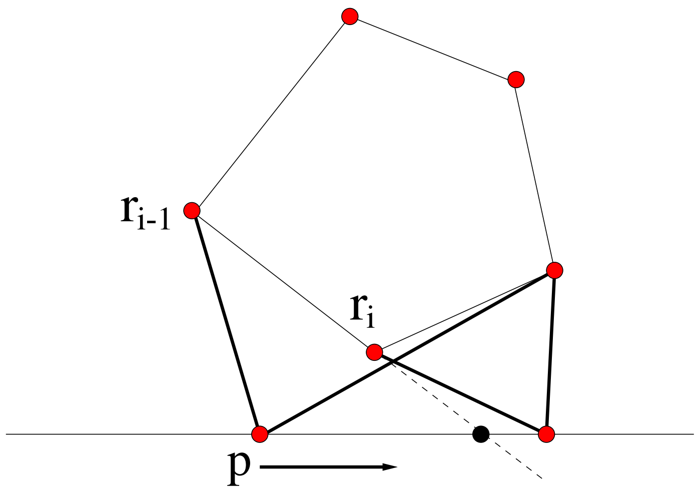


Fig. 4. A vertex of $CH(R)$ appears

r_i of $CH(\mathcal{R})$ just appears, r_i and the two adjacent vertices r_{i-1}, r_{i+1} define the largest circumcircle passing through three consecutive vertices of $CH(\mathcal{R})$. Note that in Fig. 4, r_{i+1} is the mobile red point p. As a result, r_{i+1}, r_i and r_{i-1} define a Delaunay triangle, and r_i will not contribute to the rest of the triangulation [19]. That is, the remaining triangulation remains the same and the Delaunay triangulation can be updated by adding the Delaunay triangle defined by r_{i+1}, r_i and r_{i-1} to the current triangulation. Similarly, when a vertex r_i on $CH(R)$ disappears as p moves, we update the Delaunay triangulation by deleting r_i from $CH(R)$ and remove the Delaunay triangle defined by r_i, r_{i-1} and r_{i+1} from the current Delaunay triangulation. $\square$

Case 2 instant events: These are time instants when a Voronoi edge appears or disappears from $FVD(R)$.

Lemma 8. *There are at most $O(n)$ case 2) instant events, which can be found in $O(n \log n)$ time.*

Proof. We turn our attention to the dual graph of $FVD(\mathcal{R})$, the Delaunay triangulation $DT(\mathcal{R})$. The topology of $DT(\mathcal{R})$ could change when four red points, including the mobile red point p are cocircular and the circumcircle defined by these four points encloses all red points. More specifically, a case 2 instant event is formed when p enters/leaves a circumcircle, which passes through three fixed red points and encloses all fixed red points (see Fig. 5). Observe that the center of each such enclosing circle defines a vertex of the farthest point Voronoi diagram of fixed red points $FVD(\mathcal{R} \setminus p)$, so there are $O(n)$ such enclosing circles. It takes $O(n \log n)$ time to compute $FVD(\mathcal{R} \setminus p)$. The result follows. $\square$

Lemma 9. *For each case 2 instant event, the topology of $FVD(\mathcal{R})$ can be updated in constant time.*

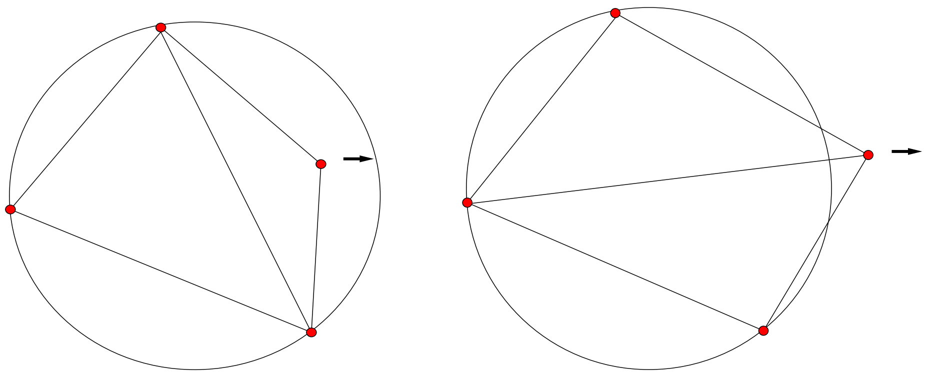


Fig. 5. Update of Delaunay triangulation of four points by edge swapping

Proof. We turn our attention to the Delaunay triangulation $DT(\mathcal{R})$ of $\mathcal{R}$. Let $\triangle(p, r_i, r_j)$ and $\triangle(r_k, r_i, r_j)$ be two adjacent triangles, which share a common edge $r_i r_j$ in $DT(\mathcal{R})$. The circumcircle $C(r_k, r_i, r_j)$ of $\triangle(r_k, r_i, r_j)$ must enclose all red points. If p leaves $C(r_k, r_i, r_j)$, $\triangle(r_k, r_i, r_j)$ is no longer a valid triangle in $DT(\mathcal{R})$. $DT(\mathcal{R})$ can be updated by edge swapping, i.e. removing edge $r_i r_j$ and adding edge pr_k. □

Case 3 instant events: These are the time instants when an exit event point moves to another edge of $VD(R)$ or appears/disappares.

Lemma 10. *Each Case 3 instant event can be identified in constant time.*

Proof. Let e be a Voronoi edge defined by p and r_i. Without loss of generality, assume e is unbounded. Let the finite end point of e be a, defined by red points points p, r_i, r_j. The exit region of e is bounded by the line segment pr_i and circumcircle $C(p, r_i, r_j)$. The exit event point of a blue point $b \in B$ lies on e if b lies in the interior of the exit region of e. The exit event point will move to another edge if and only if $C(p, r_i, r_j)$ excludes b, i.e. when p, r_i, r_j and b are curricular. This event can be determined in constant time. □

Lemma 11. *The upper bound on the number of case 3 instant events is $O(nm)$.*

Proof. Following the previous lemma, an exit event point will move to another Voronoi edge only if its corresponding blue point is cocircular with three red points which define a Voronoi vertex. Given the fact that there are $O(n)$ case 1) and case 2) instant events, there are at most $O(n)$ new Delauney triangles formed over time. Since the center of the circumcircle of each Delauney triangle defines a vertex of $FVD(\mathcal{R})$, each exit event point cannot cross more than a linear number of vertices. □

Case 4 instant events: These are the time when the candidate separating circle encloses/excludes a blue point.

Lemma 12. *Each exit event point can generate at most $O(mn)$ case 4 instant events and there are a total of $O(m^2n)$ case 4 instant events.*

Proof. Let e be a Voronoi edge defined by a fixed red point r_i and the mobile red point p, and assume the exit event point of a blue point $b \in B$ lies on e. The candidate separating circle centering at the exit event point of b is the circumcircle of r_i, p, and b. A case 4 instant event is triggered if r_i, p, and b are cocircular with another blue point. Hence, when an exit event point travels along a Voronoi edge, it can trigger at most $O(m)$ case 4 instant events.

As shown in Lemma 11, each exit event point can traverse $O(n)$ Voronoi edges. Thus, there are at most $O(m^2n)$ case 4 instant events. □

Once the trajectory of the exit event point is known, we compute the function $f_b(t)$ of the square radius of the corresponding candidate separating circle.

Following Lemma 11, each exit event point cannot cross more than $O(n)$ Voronoi edges. For a time interval $[t_{init}, t_{end}]$, $f_b(t)$ consists of $O(n)$ pieces of curves or horizontal line segments if the exit event point is stationary.

Plotting all functions $f_b(t)$ for $t \in [t_{init}, t_{end}]$ and $b \in \mathcal{B}$ on the same coordinate system gives us an arrangement H of curves of complexity $O(nm^2)$, since all functions are x-monotone and two such functions intersect no more than $O(n)$ times. H gives us the relative size between all candidate circles over time.

However, we also need to consider the blue points enclosed by the candidate circles. We further decompose H by dividing each function $f_b(t)$ at every case 4 instant event t_o generated by b by introducing a vertex at $(t_o, f_b(t_o))$. As a result, each portion of $f_b(t)$ on the new arrangement H' represents the square radius of the candidate circle for a time interval during which the blue points enclosed by the circle remain the same. The new arrangement H' has complexity $O(nm^2)$. We call each portion of $f_b(t)$ on H' a *simple curve.*

Let the blue point count of a simple curve on H' be the number of blue points enclosed by the corresponding candidate circle. The last step to compute the locus of the minimum separating circle is to extract the lower envelope of curves with the lowest blue point count. Each curve on the lower envelope gives us the minimum separating circle for the interval spanned by the curve, hence, the locus of the center of the minimum separating circle.

We use a plane sweep to extract such lower envelope. We sweep H' by a vertical line. At any moment, the sweep line intersects with at most m functions $f_b(t)$, for $b \in \mathcal{B}$. Each function is indexed by its blue point count at the current moment. We build a hash table for functions intersected by the sweep line. For each entry of the hash table, all functions are maintained in a balanced tree by the order intersected by the sweep line. Note that we only have to update the hash table at vertices of H'. If two functions with the same blue point count intersect, we need to exchange their position in the corresponding tree. If the sweep line crosses a vertex introduced by a case 4 instant event, the blue point count of a function changes. The corresponding function will be moved to the appropriate entry of the hash table. It takes $O(\log m)$ time to update the hash table for each instance.

Theorem 2. *The locus of the center of the minimum separating circle has complexity $O(nm^2)$ and can be computed in $O(nm^2 \log m)$ time.*

6 The Minimum Separating Circle with Multiple Moving Points

In this section, we discuss the general version of the kinetic red-blue separating circle problem. That is, given a set $\mathcal{R}$ of n red points and a set $\mathcal{B}$ of m blue points, where each red or blue point is either stationary or moves with constant speed along a straight line, we want to find the trajectory of the center of the minimum separating circle.

We need to maintain the following kinetic data structures: kinetic convex hull of $\mathcal{R}$, kinetic farthest point Voronoi diagram of $\mathcal{R}$, trajectories of exit event points, and blue point counts for candidate circles.

We apply data structure proposed in [4] to maintain the kinetic convex hull of $\mathcal{R}$. In [4], the problem of maintaining the convex hull of moving points is dualized by mapping each point to a line in the dual plane using the standard point-line duality transform and computing the upper and lower envelopes of the family of dual lines. The data structure uses $O(n)$ certificates that involves comparisons between line slopes, and a balanced binary tree to store the combinatorial description of the convex hull. At most $O(\log n)$ certificates are updated when an event is triggered. Hence, it takes $O(\log n)$ time to process each event. The data structure generates at most $O(n^{2+\epsilon})$ events.

Similar to $P2$, in order to maintain $FVD(\mathcal{R})$, we define two classes of instant events:

case 1) the appearance/disappearance of a vertex on the boundary of the convex hull $CH(R)$ of $\mathcal{R}$,
case 2) the appearance/disappearance of a vertex on the farthest neighbor Voronoi diagram $FVD(\mathcal{R})$ of $\mathcal{R}$.

Lemma 13. *We can update $FVD(\mathcal{R})$ in constant time when a case 1 or a case 3 event occurs and in the worst case there are $O(n^{3+\epsilon})$ events.*

Proof. We turn our attention to the dual graph of $FVD(\mathcal{R})$, the Delaunay triangulation $DT(\mathcal{R})$, and show how to maintain $DT(\mathcal{R})$ instead, since it is more nature and easier to describe the update procedure in the dual setting. For each case 1 instant event, $DT(\mathcal{R})$ can be updated in constant time by introducing a new triangle to the original triangulation as described in Lemma. 7. There are $O(n^{2+\epsilon})$ such events. For each case 2 instant event, $DT(\mathcal{R})$ can also be updated in constant time by edge swapping as described in Lemma. 9. A case 2 event will occur when four red points are cocircular and the circumcircle formed by these four points encloses all other red points. By the fact that four points moving with constant speed along straight lines can be cocircular at most four times, we have a $O(n^4)$ trivial upper bound on the number of case 2 event points. However, by a Davenport-Schinzel sequence sequence argument, Albers et. al. [1] shows that there are at most $O(n^{3+\epsilon})$ such events [1]. $\square$

To update the trajectory of each event point, we need to identify the following class of instant events:
case 3) some exit event point associated with a blue point moves to another Voronoi edge.

Lemma 14. *There are at most $O(mn^{2+\epsilon})$ case (3) instant events.*

Proof. A case 3 instant event will occur when some exit event passes through a Voronoi vertex, i.e. when some blue point and three red points are cocircular and the circumcircle of the three red points encloses $\mathcal{R}$. For a blue point b and one red point r, it is shown in [1] that number of enclosing circles of $\mathcal{R}$ which pass through b, r and two red other points is $O(n^{1+\epsilon})$. Considering all pairs of one red point and one blue point, we obtain the upper bound on the number of case 3 instant events. □

Lemma 15. *For each case 3 instant event, it takes $O(n)$ time to update the trajectory of the corresponding exit event point.*

Proof. Since we cannot assume all point are in general position at each instant event, the Voronoi vertex was just passed by the exit event point could have degree $O(n)$. Thus, we need to check $O(n)$ exit regions associated with Voronoi edges incident to this vertex. □

To track the count of blue points enclosed by each candidate circle, we define another class of instant events:
case 4) the candidate separating circle associated with an exit event point encloses/excludes a blue point.

Lemma 16. *There are at most $O(m^2n^{1+\epsilon})$ case 4) instant events.*

Proof. Recall that a candidate separating circle is a circle that passes through two red points and one blue point and encloses $\mathcal{R}$. Hence, a case 4 event will be triggered when two blue points and two red points are cocircular and their circumcircle encloses all red points. Following the same argument as in Lemma. 14, we can show that each pair of blue points generates $O(n^{1+\epsilon})$ case 4 instant events. □

We apply the approach proposed in Section 5 to compute the locus of the center of the minimum separation circle. As shown in the proof of Lemma 14, one blue point can generate at most $O(n^{2+\epsilon})$ case 3 event points. The function $f_b(t)$ of the square radius of the candidate separating circle of a blue point $b \in \mathcal{B}$ consists of $O(n^{2+\epsilon})$ curves. Hence, the arrange H of all such functions has a complexity of $O(m^2n^{2+\epsilon})$. After introducing $O(m^2n^{1+\epsilon})$ additional vertices due to case 4 instant events, the final arrangement H' has a complexity of $m^2n^{2+\epsilon}$. It takes a total of $O(mn^{3+\epsilon})$ time to maintain trajectories of all exit event points.

Theorem 3. *The locus of the center of the minimum separating circle has complexity $O(m^2n^{2+\epsilon})$ and can be found in $O(m^2n^{2+\epsilon}\log(mn) + mn^{3+\epsilon})$ time.*

References

1. Albers, G., Guibas, L.J., Mitchell, J.S.B., Roos, T.: Voronoi diagrams of moving points. Int. J. Comput. Geometry Appl. 8(3), 365–380 (1998)
2. Aronov, B., Rappaport, D., Seara, C., Garijo, D., Núñez, Y., Urrutia, J.: Measuring the error of linear separators on linearly inseparable data. In: XIII Encuentros de Geometria Computacional, Zaragoza, España (2009)
3. Atallah, M.J.: Dynamic computational geometry (preliminary version). In: FOCS, pp. 92–99 (1983)
4. Basch, J., Guibas, L.J., Hershberger, J.: Data structures for mobile data. J. Algorithms 31(1), 1–28 (1999)
5. Basch, J., Guibas, L.J., Silverstein, C., Zhang, L.: A practical evaluation of kinetic data structures. In: Symposium on Computational Geometry, pp. 388–390 (1997)
6. Bitner, S., Cheung, Y.K., Daescu, O.: Minimum separating circle for bichromatic points in the plane. In: ISVD, pp. 50–55 (2010)
7. Demain, E., Einsenstat, S., Guibas, L., Schulz, A.: Kinetic minimum spanning circle. In: Proceedings of the Fall Workshop on Computational Geometry, New York, USA (2010)
8. Mitchell, J.S.B., Seara, C., Arkin, E.M., Hurtado, F., Skiena, S.: Some lower bounds on geometric separability problems. Int. J. Comput. Geometry Appl. 16(1), 1–26 (2006)
9. Seara, C., Hurtado, F., Sethia, S.: Red-blue separability problems in 3d. International Journal of Computational Geometry 15(2), 167–192 (2005)
10. Fekete, S.: On the complexity of min-link red-blue separation (1992) (manuscript)
11. Fisk, S.: Separating point sets by circles, and the recognition of digital disks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 554–556 (July 1986)
12. Guibas, L.J.: Kinetic data structures: a state of the art report. In: WAFR 1998 (1998)
13. Hurtado, F., Mora, M., Ramos, P.A., Seara, C.: Separability by two lines and by nearly straight polygonal chains. Discrete Applied Mathematics 144(1-2), 110–122 (2004)
14. Hurtado, F., Noy, M., Ramos, P.A., Seara, C.: Separating objects in the plane by wedges and strips. Discrete Applied Mathematics 109(1-2), 109–138 (2001)
15. Megiddo, N.: Linear-time algorithms for linear programming in R^3 and related problems. SIAM Journal on Computing 12(4), 759–776 (1983)
16. O'Rourke, J., Kosaraju, S., Megiddo, N.: Computing circular separability. Discrete Computational Geometry 1, 105–113 (1986)
17. Rahmati, Z., Zarei, A.: Combinatorial changes of euclidean minimum spanning tree of moving points in the plane. In: CCCG, pp. 43–45 (2010)
18. Roos, T.: Voronoi diagrams over dynamic scenes. Discrete Applied Mathematics 43(3), 243–259 (1993)
19. Skyum, S.: A simple algorithm for computing the smallest enclosing circle. Information Processing Letters 37(3) (1991)

A Semantic Model for Many-Core Parallel Computing*

Nan Zhang and Zhenhua Duan**

Institute of Computing Theory and Technology, and ISN Laboratory
Xidian University, Xi'an 710071, P.R. China
nanzhang@stu.xidian.edu.cn, zhhduan@mail.xidian.edu.cn

Abstract. Many-core parallel computing and programming is a new challenge for formal specification and verification. This paper presents a semantic model for many-core parallel computing systems so that the systems can be modeled and verified in a manageable way. The model is called Cylinder Computation Model (CCM) which is based on projection constructs in Projection Temporal Logic (PTL) and Modeling, Simulation and Verification Language (MSVL). To this end, the syntax and semantics of CCM are presented in details. Further, some logic laws regarding CCM are given and the normal form of CCM programs is formalized and proved. Moreover, the operational semantics of CCM and an algorithm for implementing CCM programs with MSVL are also formalized. Finally, an example, simple word processor, is given to show how CCM works under MSVL paradigm.

Keywords: Many-core, Parallel Computing, Temporal Logic, Projection, Model.

1 Introduction

A parallel program consists of more processes and each process is a sequential program with its own local variables and statements. Parallel programs are more difficult to deal with than sequential ones. Creating a correct parallel program is not a straightforward process even for a considerable small problem because programmers are forced to consider that the program will always yield to a correct result no matter what order the instructions are executed in. Thus for a parallel program, testing is not able to convince us the correctness of the program. So, to ensure the correctness of parallel programs, formal verification is an important viable solution [16,18].

Temporal logic has gained recognition as a competent and versatile formalism for rigorously specifying and verifying concurrent or parallel systems such as hardware circuits, safety critical systems and communication protocols. In particular, Projection Temporal Logic (PTL) [6], an extension of Interval Temporal Logic (ITL)[17], and its executable subset called Modeling, Simulation and Verification Language (MSVL)[23] are useful formalisms for modeling and verifying parallel or concurrent systems.

* This research is supported by the NSFC Grant No.60910004, 60873018, 91018010, 61003078, and 61003079, 973 Program Grant No.2010CB328102, SRFDP Grant No.200807010012 and ISN Lab Grant No.ISN1102001.

** Corresponding author.

The contribution of this paper is to give a more flexible parallel model in temporal logic programming so that it can be used to handle many-core parallel computing. As a result, we present a semantic model based on MSVL for many-core parallel computing, namely Cylinder Computation Model (CCM). CCM has a typical form of $\phi_1 \texttt{ ov } (l_1) \| \ldots \| \phi_m \texttt{ ov } (l_m)$ where $\phi_i (i = 1..m)$ are MSVL programs. With this parallelism, a main time interval is the sequence of fine-grained unit subintervals with length one while several coarse-grained projected intervals over which processes are interpreted are in parallel with the main time interval. This computation model can be viewed as m processes that share one chip and each occupies a processor respectively cooperating to solve a problem in a parallel way. Each process progresses in its own speed and communicates with each other at some global states. Sequence expression l_i is used to control and determine the execution of ϕ_i. As you can see, parallel construct ($\|$) is the main operator in CCM. Thus $\phi_1 \texttt{ ov } (l_1) \| \ldots \| \phi_m \texttt{ ov } (l_m)$ is endowed with a meaning of many-core. As we mentioned earlier, programs are interpreted over a state sequence. In each new state, processes deployed on different cores do their own job, read data from memory, compute or update memory. As an example, a simple word processor is given to illustrate how CCM can be used to specify many-core parallel programs. To implement CCM programs in MSVL, we investigate the operational semantics of CCM. Further, an interpreter has been developed based on the operational semantics.

The paper is organized as follows: the syntax and semantics of PTL are described in next section; MSVL is also introduced in section 2, including framing technology, expressions and statements, normal form of MSVL. CCM programs with MSVL are defined in section 3 and 4, including syntax and semantics as well as operational semantics. In section 5, the implementation of CCM is illustrated within the interpreter of MSVL. Further, an example is given to show the practicability of CCM in parallel programming. In addition, some work related to models of parallel computation has been discussed. Finally, conclusions are drawn in section 8.

2 Preliminaries

2.1 Projection Temporal Logic

Our underlying logic is Projection Temporal Logic. In the following, we briefly introduce its syntax and semantics. The detail can be found in [6].

Syntax. Let $Prop$ be a countable set of propositions, and V a countable set of typed variables consisting of static and dynamic variables. B represents the boolean domain, D denotes all data needed including integers, lists, sets etc. N_0 stands for non-negative integers and $N_0^+ = N_0 - \{0\}$. Terms e and formulas ϕ are defined as follows:

$$\begin{aligned} e ::= {} & v \mid \bigcirc e \mid \odot e \mid \mathsf{beg}(e) \mid \mathsf{end}(e) \mid f(e_1, \ldots, e_n) \\ \phi ::= {} & p \mid e_1 = e_2 \mid P(e_1, \ldots, e_n) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x : \phi \mid \bigcirc \phi \\ & \mid (\phi_1, \ldots, \phi_m)\, prj\ \phi \mid (\phi_1, \ldots, (\phi_i, \ldots, \phi_l)^{\oplus}, \ldots, \phi_m)\, prj\ \phi \end{aligned}$$

where $v \in V$ and $p \in Prop$. A formula (term) is called a state formula (term) if it contains no temporal operators, otherwise it is a temporal formula (term).

Semantics. A state s over $V \cup Prop$ is defined to be a pair (I_v, I_p) of state interpretations I_v and I_p. I_v assigns each variable $v \in V$ a value in D or nil (undefined) and the total domain is denoted by $D' = D \cup \{nil\}$, whereas I_p assigns each proposition $p \in Prop$ a truth value in B. $s[v]$ denotes the value of v at state s.

An interval σ is a non-empty sequence of states, which can be finite or infinite. The length, $|\sigma|$, of σ is ω if σ is infinite, and the number of states minus 1 if σ is finite. We extend the set N_0 of non-negative integers to include ω, denoted by $N_\omega = N_0 \cup \{\omega\}$ and extend the comparison operators, $=, <, \leq$, to N_ω by considering $\omega = \omega$, and for all $i \in N_0$, $i < \omega$. Furthermore, we define $\preceq$ as $\leq -\{(\omega, \omega)\}$. For concise of presentation, $\langle s_0, \ldots, s_{|\sigma|}\rangle$ is denoted by σ, where $s_{|\sigma|}$ is undefined if σ is infinite. The concatenation of a finite σ with another interval (or empty string) $\sigma^{'}$ is denoted by $\sigma \bullet \sigma^{'}$ (not sharing any states). Let $\sigma = \langle s_0, s_1, \ldots, s_{|\sigma|}\rangle$ be an interval and $r_1, \ldots, r_h$ be integers $(h \geq 1)$ such that $0 \leq r_1 \leq r_2 \leq \ldots \leq r_h \preceq |\sigma|$. The projection of σ onto $r_1, \ldots, r_h$ is the interval (called projected interval) $\sigma \downarrow (r_1, \ldots, r_h) = \langle s_{t_1}, s_{t_2}, \ldots, s_{t_l}\rangle$ where $t_1, \ldots, t_l$ are obtained from $r_1, \ldots, r_h$ by deleting all duplicates. That is, $t_1, \ldots, t_l$ is the longest strictly increasing subsequence of $r_1, \ldots, r_h$. For instance, $\langle s_0, s_1, s_2, s_3, s_4\rangle \downarrow (0, 0, 2, 2, 2, 3) = \langle s_0, s_2, s_3\rangle$. We also need to generalize the notation of $\sigma \downarrow (r_1, \ldots, r_h)$ to allow r_i to be ω. For an interval $\sigma = \langle s_0, s_1, \ldots, s_{|\sigma|}\rangle$ and $0 \leq r_1 \leq \ldots \leq r_h \leq |\sigma|$ $(r_i \in N_\omega)$, we define $\sigma \downarrow (r_1, \ldots, r_h, \omega) = \sigma \downarrow (r_1, \ldots, r_h)$. It is assumed that a static variable remains the same over an interval whereas a dynamic variable can have different values at different states. To evaluate the existential quantification, an equivalence relation is required [6] and given below. We use I_v^k and I_p^k to denote the state interpretations at state s_k.

Definition 1 (**x-equivalence**)**.** *Two intervals, σ and σ', are x-equivalent, denoted by $\sigma' \overset{x}{=} \sigma$, if $|\sigma| = |\sigma'|$, $I_v^h[y] = I_v^{'h}[y]$ for all $y \in V - \{x\}$, and $I_p^h[p] = I_p^{'h}[p]$ for all $p \in Prop$ $(0 \leq h \preceq |\sigma|)$.*

An interpretation is a quadruple $\mathcal{I} = (\sigma, i, k, j)$, where σ is an interval, $i, k \in N_0$, and $j \in N_\omega$ such that $0 \leq i \leq k \preceq j \leq |\sigma|$. We use the notation (σ, i, k, j) to indicate that some formula ϕ or term e is interpreted over the subinterval $\langle s_i, \ldots, s_j\rangle$ of σ with the current state being s_k. For every term e, the evaluation of e relative to interpretation $\mathcal{I} = (\sigma, i, k, j)$, denoted by $\mathcal{I}[e]$, is defined by induction on terms as follows:

$$
\begin{array}{lll}
1.\ \mathcal{I}[v] & = & \begin{cases} s_k[v] = I_v^k[v] = I_v^i[v] & \text{if v is a static variable.} \\ s_k[v] = I_v^k[v] & \text{if v is a dynamic variable.} \end{cases} \\
2.\ \mathcal{I}[\bigcirc e] & = & \begin{cases} (\sigma, i, k+1, j)[e] & \text{if } k < j \\ nil & \text{otherwise} \end{cases} \\
3.\ \mathcal{I}[\circleddash e] & = & \begin{cases} (\sigma, i, k-1, j)[e] & \text{if } i < k \\ nil & \text{otherwise} \end{cases} \\
4.\ \mathcal{I}[\mathsf{beg}(e)] & = & (\sigma, i, i, j)[e] \\
5.\ \mathcal{I}[\mathsf{end}(e)] & = & \begin{cases} (\sigma, i, j, j)[e] & \text{if } j < \omega \\ nil & \text{otherwise} \end{cases} \\
6.\ \mathcal{I}[f(e_1, \ldots, e_n)] & = & \begin{cases} nil & \text{iff } \mathcal{I}[e_h] = nil, \text{ for some } h \in \{1..n\} \\ \mathcal{I}[f](\mathcal{I}[e_1], \ldots, \mathcal{I}[e_n]) & \text{otherwise} \end{cases}
\end{array}
$$

The meaning of formulas is given by the satisfaction relation, $\models$, which is inductively defined as follows:

1. $\mathcal{I} \models p$ iff $I_p^k[p] = true$, for any given proposition p.
2. $\mathcal{I} \models P(e_1, \ldots, e_n)$ iff P is a primitive predicate other than $=$ and, for all h, $0 \leq h \leq n$, $\mathcal{I}[e_h] \neq nil$ and $\mathcal{I}[P](\mathcal{I}[e_1], \ldots, \mathcal{I}[e_n]) = true$.
3. $\mathcal{I} \models e_1 = e_2$ iff e_1 and e_2 are terms and $\mathcal{I}[e_1] = \mathcal{I}[e_2]$.
4. $\mathcal{I} \models \neg\phi$ iff $\mathcal{I} \not\models \phi$.
5. $\mathcal{I} \models \phi_1 \wedge \phi_2$ iff $\mathcal{I} \models \phi_1$ and $\mathcal{I} \models \phi_2$.
6. $\mathcal{I} \models \bigcirc\phi$ iff $k < j$ and $(\sigma, i, k+1, j) \models \phi$.
7. $\mathcal{I} \models \exists x : \phi$ iff there exists an interval σ' such that $\sigma'_{(i..j)} \overset{x}{=} \sigma_{(i..j)}$ and $(\sigma', i, k, j) \models \phi$.
8. $\mathcal{I} \models (\phi_1, \ldots, \phi_m)\ prj\ \phi$ iff there exist integers $k = r_0 \leq \cdots \leq r_{m-1} \preceq r_m \leq j$; for all $1 \leq l \leq m$, $(\sigma, i, r_{l-1}, r_l) \models \phi_l$; $\sigma' \models \phi$ for one of the following σ':
   (a) $r_m < j$ and $\sigma' = \sigma \downarrow (i, r_0, \ldots, r_m) \bullet \sigma_{(r_m+1..j)}$, or
   (b) $r_m = j$ and $\sigma' = \sigma \downarrow (i, r_0, \ldots, r_h)$ for some $0 \leq h \leq m$.
9. $\mathcal{I} \models (\phi_1, \ldots, (\phi_i, \ldots, \phi_l)^{\oplus}, \ldots, \phi_m)\ prj\ \phi$ iff one of following cases holds:
   (a) $1 \leq i \leq l \leq m$ and there exists an integer $n \geq 1$ and $\mathcal{I} \models (\phi_1, \ldots, (\phi_i, \ldots, \phi_l)^{(n)}, \ldots, \phi_m)\ prj\ \phi$, or
   (b) $1 \leq i \leq l = m$, $j = \omega$ and there exist infinitely many integers $k = r_0 \leq r_1 \leq \cdots \leq r_n \preceq \omega$ and $\lim_{n\to\infty} r_n = \omega$ such that for all $1 \leq x \leq i-1$, $(\sigma, i, r_{x-1}, r_x) \models \phi_x$, and $(\sigma, i, r_{i+t(l-i+1)+n-1}, r_{i+t(l-i+1)+n}) \models \phi_{i+n}$, for all $t \geq 0$ and $0 \leq n \leq l-i$, and $\sigma \downarrow (i, r_0, r_1, \ldots, r_h, \omega) \models \phi$ for some $h \in N_\omega$.

The abbreviations true, false, $\vee$, $\rightarrow$ and $\leftrightarrow$ are defined as usual. In particular, true $\overset{\text{def}}{=} \phi \vee \neg\phi$ and false $\overset{\text{def}}{=} \phi \wedge \neg\phi$ for any formula ϕ. We also use the following abbreviations:

A1	ε	$\overset{\text{def}}{=} \neg\bigcirc\text{true}$	A2	more	$\overset{\text{def}}{=} \neg\varepsilon$
A3	$\bigcirc^0\phi$	$\overset{\text{def}}{=} \phi$	A4	$\bigcirc^n\phi$	$\overset{\text{def}}{=} \bigcirc(\bigcirc^{n-1}\phi)\ (n > 0)$
A5	$\odot\phi$	$\overset{\text{def}}{=} \varepsilon \vee \bigcirc\phi$	A6	$\phi_1; \phi_2$	$\overset{\text{def}}{=} (\phi_1, \phi_2)\ prj\ \varepsilon$
A7	$\Diamond\phi$	$\overset{\text{def}}{=} \text{true}; \phi$	A8	$\Box\phi$	$\overset{\text{def}}{=} \neg\Diamond\neg\phi$
A9	$\text{len}(n)$	$\overset{\text{def}}{=} \bigcirc^n\varepsilon$	A10	skip	$\overset{\text{def}}{=} \text{len}(1)$
A11	$\forall x : \phi$	$\overset{\text{def}}{=} \neg\exists x : \neg\phi$	A12	$\phi_1 \| \phi_2$	$\overset{\text{def}}{=} \phi_1 \wedge (\phi_2; \text{true}) \vee (\phi_1; \text{true}) \wedge \phi_2$

2.2 Modeling, Simulation and Verification Language

MSVL extends Tempura [17] to include infinite models, a new projection construct, a previous operator, framing constructs, a synchronization construct await(c), and non-deterministic statements [5,6,7,23].

Framing. Framing is concerned with how the value of a variable can be carried from one state to the next. To give the frame operator, we define assignment operators firstly.

$$x \Leftarrow e \overset{\text{def}}{=} x = e \wedge p_x$$
$$x \,\circ\!\!=^{\dagger} e \overset{\text{def}}{=} \exists a : (a = e \wedge \bigcirc(x \Leftarrow a))$$
$$x :=^{\dagger} e \overset{\text{def}}{=} x \,\circ\!\!=^{\dagger} e \wedge \text{skip}$$

where a is a static variable, e an expression, p_x an atomic proposition associated with dynamic variable x and cannot be used for other purpose. To identify an occurrence of an assignment to a variable x, we make use of a flag called *the assignment flag*, denoted by a predicate af(x) which can be defined as af$(x) \overset{\text{def}}{=} p_x$ under the minimal model semantics [6,7]. Whenever an assignment to x is encountered, $\mathsf{af}(x)$ should be true. Then frame operators are defined as follows: Let b be a static variable, and $x_1, \ldots, x_k$ dynamic variables.

$$\mathsf{lbf}(x) \overset{\text{def}}{=} \neg\mathsf{af}(x) \to \exists b : (\odot x = b \wedge x = b)$$
$$\mathsf{frame}(x) \overset{\text{def}}{=} \Box(\mathsf{more} \to \bigcirc\mathsf{lbf}(x))$$
$$\mathsf{frame}(x_1, \ldots, x_k) \overset{\text{def}}{=} \mathsf{frame}(x_1) \wedge \ldots \wedge \mathsf{frame}(x_k)$$

MSVL Programs. With MSVL, expressions can be treated as terms and statements can be treated as formulas in PTL. The arithmetic and boolean expressions of MSVL can be inductively defined as follows [6,7]:

$$e ::= n \mid x \mid \bigcirc x \mid \odot x \mid e_0 \; op \; e_1 \quad (op ::= + \mid - \mid \times \mid mod)$$
$$b ::= \mathsf{true} \mid \mathsf{false} \mid \neg b \mid b_0 \wedge b_1 \mid e_0 = e_1 \mid e_0 < e_1$$

where n is an integer and x is a static or dynamic variable. One may refer to the value of a variable at the previous state or the next state. The statements of MSVL programs can be inductively defined by the following grammar [6,7]:

1.Termination: ε
2.Assignment: $x = e$
3.Positive Immediate Assignment: $x \Leftarrow e$
4.State Frame: $\mathsf{lbf}(x)$
5.Interval Frame: $\mathsf{frame}(x)$
6.Conjunction: ϕ_0 `and` ϕ_1
7.Selection: ϕ_0 or ϕ_1
8.Projection: $(\phi_1, \ldots, \phi_m) \; prj \; \phi$
9.Next: `next` ϕ
10.Always: `always` ϕ
11.Conditional: `if` b `then` ϕ_0 `else` ϕ_1
12.Existential Quantification: $\exists x : \phi(x)$
13.Sequential: ϕ_0 ; ϕ_1
14.While: `while` b `do` ϕ
15.Parallel: $\phi_0 \| \phi_1$
16.Synchronous Communication: $\mathsf{await}(c)$

To synchronize communication of parallel processes in a parallel program, a synchronization construct, $\mathsf{await}(c)$ is required and $\mathsf{await}(c) \overset{\text{def}}{=} \mathsf{frame}(x_1, \ldots, x_n) \wedge \mathsf{halt}(c)$. The $\mathsf{await}(c)$ does not change any variable, but waits until the condition c becomes true, at which point it terminates. $x_1, \ldots, x_n$ are dynamic variables appeared in c.

Normal form. Since temporal logic formulas are interpreted over intervals and temporal logic programming language is a subset of one corresponding temporal logic, programs should be executed over a sequence of states which is a marked feature of temporal logic programming. So execution of programs is implemented by reduction since a program can be reduced to two parts: present and remain. That fact can be illustrated by the normal form of programs given in [23].

Definition 2 (normal form of MSVL). *An MSVL program ϕ is in normal form if*

$$\phi \overset{\text{def}}{=} (\bigvee_{i=1}^{k} \phi_{ei} \wedge \varepsilon) \vee (\bigvee_{j=1}^{h} \phi_{cj} \wedge \bigcirc\phi_{fj})$$

where $k + h \geq 1$ and the following hold:

- *ϕ_{fj} is an internal program in which variables may refer to the previous states but not beyond the first state of the current interval over which the program is executed.*
- *each ϕ_{ei} and ϕ_{cj} is either* true *or a state formula of the form $p_1 \wedge \ldots \wedge p_m$ $(m \geq 1)$ and each p_l $(1 \leq l \leq m)$ is either $(x = e)$ with $e \in D$, $x \in V$, or p_x, or $\neg p_x$.*

Theorem 1. *For each MSVL program ϕ there is a program ϕ' in normal form satisfying $\phi \equiv \phi'$.*

Theorem 1 tells us that for each MSVL program there is an equivalent program in normal form. The proof of the theorem can be found in [7].

3 Cylinder Computation Model

A semantic model for parallel computations called Cylinder Computation Model is proposed based on MSVL in this paper. The model has the following typical form:

$$\phi_1 \ \texttt{ov} \ (n_1, \ldots, n_{m_1}) \parallel \phi_2 \ \texttt{ov} \ (n'_1, \ldots, n'_{m_2})$$

In the context ϕ_1 and ϕ_2 are MSVL programs, $n_i (i = 1..m_1)$ and $n'_j (j = 1..m_2)$ are positive integers, and $\parallel$ is the parallel operator in MSVL. As a matter of fact, the above statement is an abbreviation of the MSVL statement:

$$(\mathsf{len}(n_1), \mathsf{len}(n_2), \ldots, \mathsf{len}(n_{m_1}))\, prj\ \phi_1 \| (\mathsf{len}(n'_1), \mathsf{len}(n'_2), \ldots, \mathsf{len}(n'_{m_2}))\, prj\ \phi_2$$

in which ϕ_1 and ϕ_2 are executed in parallel and share some specified states for communication. These states are specified by n_i and n'_j. With this semantics, we can use the sequence of $\mathsf{len}(x)$'s to control a specific execution of program ϕ. We can make use of the statement to model the execution of several processes deployed over a variety of cores on one chip. As you can see, processes $\phi_1, \ldots$, and ϕ_i are executed over different time granularity. For example, the interval satisfying a parallel program $\phi_1 \ \texttt{ov} \ (2, 3, 3, 4) \parallel \phi_2 \ \texttt{ov} \ (3, 5, 3, 6) \parallel \phi_3 \ \texttt{ov} \ (2, 1, 2, 3, 3, 1, 5)$ is given in Figure 1. Here ϕ_1, ϕ_2 and ϕ_3 are executed over three cores on one chip. They are autonomous and communicate only at some specified time points. If two projected intervals share a projected state, the two processes communicate at the state. It is possible that at a moment one process updates the value of a variable, while the other process accesses the variable. Thus the latter process obtains the new value of the variable, and the communication between processes is done.

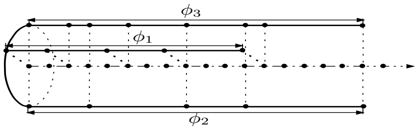


Fig. 1. Interpretation of $\phi_1 \ \texttt{ov} \ (2, 3, 3, 4) \parallel \phi_2 \ \texttt{ov} \ (3, 5, 3, 6) \parallel \phi_3 \ \texttt{ov} \ (2, 1, 2, 3, 3, 1, 5)$

Syntax of sequence expressions. $\quad l ::= n \mid l_1 + l_2 \mid l_1, l_2 \mid l^{\circledast} \ \ (n \in N_0^+)$

A *sequence expression* is a sequence of positive integers. We use L to denote the set of all sequence expressions. The semantics of sequence expressions is also defined by a satisfaction relation, $\Vdash$, by means of interpretation $\mathcal{I}$.

Semantics of sequence expressions

1. $\mathcal{I} \Vdash n$ iff $j - k = n$.
2. $\mathcal{I} \Vdash l_1 + l_2$ iff $\mathcal{I} \Vdash l_1$ or $\mathcal{I} \Vdash l_2$.
3. $\mathcal{I} \Vdash l_1, l_2$ iff (a) there exists r, $k \leq r \preceq j$, such that $\mathcal{I}_1 = (\sigma, i, k, r) \Vdash l_1$ and $\mathcal{I}_2 = (\sigma, r, r, j) \Vdash l_2$, or
   (b) $j = \omega$ and $(\sigma, i, k, j) \Vdash l_1$ and l_2 is satisfiable.
4. $\mathcal{I} \Vdash l^{\circledast}$ iff (a) there exist finitely many integers $k = r_0 \leq r_1 \leq \ldots \preceq r_n = j$ such that for all h, $1 \leq h \leq n$, $(\sigma, r_{h-1}, r_{h-1}, r_h) \Vdash l$, or
   (b) there exist infinitely many integers $k = r_0 \leq r_1 \leq \ldots$, $\lim_{n \to \infty} r_n = \omega$ such that for all $h \geq 1$, $(\sigma, r_{h-1}, r_{h-1}, r_h) \Vdash l$.

With CCM programs, the execution of ϕ is controlled by the sequence expressions l. The beginning and ending points of the execution of each n in l make up of the coarse-grain interval of ϕ. Therefore, to give the semantics of CCM programs, it is necessary to define the set of ending point lists denoted by $S_l^{\mathcal{I}}$.

Definition 3 (set of ending point lists $S_l^{\mathcal{I}}$). *Let $\mathcal{I}$ be an arbitrary interpretation (σ, i, k, j). $S_l^{\mathcal{I}}$ is inductively defined as follows:*

1. *If $\mathcal{I} \not\Vdash l$, then $S_l^{\mathcal{I}} = \emptyset$.*
2. *If $\mathcal{I} \Vdash n$, then $S_n^{\mathcal{I}} = \{\langle k, j\rangle\}$.*
3. *If $\mathcal{I} \Vdash l_1, l_2$, then* $S_{l_1,l_2}^{\mathcal{I}} = \left\{ t_1 \bullet t_2 \;\middle|\; \begin{array}{l} \text{there exists } r, k \le r \preceq j, \text{ such that } \mathcal{I}_1 = \\ (\sigma, i, k, r) \Vdash l_1 \text{ and } \mathcal{I}_2 = (\sigma, r, r, j) \Vdash l_2 \text{ and} \\ t_1 \in S_{l_1}^{\mathcal{I}_1} \text{ and } t_2 \in S_{l_2}^{\mathcal{I}_2} \end{array} \right\}$ $\bigcup \{t_1 \mid j = \omega \text{ and } l_2 \text{ is satisfiable and } \mathcal{I} \Vdash l_1 \text{ and } t_1 \in S_{l_1}^{\mathcal{I}}\}$
4. *If $\mathcal{I} \Vdash l_1 + l_2$, then $S_{l_1+l_2}^{\mathcal{I}} = S_{l_1}^{\mathcal{I}} \bigcup S_{l_2}^{\mathcal{I}}$.*
5. *If $\mathcal{I} \Vdash l^{\circledast}$, then* $S_l^{\mathcal{I}} = \left\{ \mathop{\bullet}_{h=1}^{n} t_h \;\middle|\; \begin{array}{l} \text{there exist finitely many integers } k = r_0 \le r_1 \ldots \preceq \\ r_n = j \text{ such that for all } 1 \le h \le n, \mathcal{I}_h = \\ (\sigma, r_{h-1}, r_{h-1}, r_h) \Vdash l \text{ and } t_h \in S_l^{\mathcal{I}_h} \end{array} \right\}$ $\bigcup \left\{ \mathop{\bullet}_{h=1}^{\infty} t_h \;\middle|\; \begin{array}{l} \text{there exist infinitely many integers } k = r_0 \le r_1 \le \\ \ldots, \lim_{n\to\infty} r_n = \omega \text{ such that for all } h \ge 1, \mathcal{I}_h = \\ (\sigma, r_{h-1}, r_{h-1}, r_h) \Vdash l \text{ and } t_h \in S_l^{\mathcal{I}_h} \end{array} \right\}$

The precedence rules are given from high to low: (1) $\circledast$(iteration); (2) ,(concatenation); (3) $+$(selection). In addition, we extend L to include two extra sequence expressions: $\bot$ and 0, standing for empty set and empty string respectively. As a result, the extended sequence expressions are now in fact ω regular expressions. Let $L' = L \cup \{\bot, 0\}$ and $l, l_1, l_2, l_3 \in L$, we have the following properties.

R1. $l_1, (l_2 + l_3) = l_1, l_2 + l_1, l_3$	R2. $\bot, l = l, \bot = \bot$	R3. $l_1 + l_2 = l_2 + l_1$
R4. $(l_1 + l_2), l_3 = l_1, l_3 + l_2, l_3$	R5. $0, l = l, 0 = l$	R6. $l + l = l$
R7. $l_1, (l_2, l_3) = (l_1, l_2), l_3$	R8. $l, l^{\circledast} = l^{\circledast}, l$	R9. $l^{\circledast} = l + l, l^{\circledast}$
R10. $l_1, (l_2, l_1)^{\circledast} = (l_1, l_2)^{\circledast}, l_1$	R11. $l^{\circledast}, l^{\circledast} = l, l^{\circledast}$	R12. $(l^{\circledast})^{\circledast} = l^{\circledast}$
R13. $l_1 + (l_2 + l_3) = (l_1 + l_2) + l_3$		

Based on sequence expressions, the syntax and semantics of CCM can be defined.

Syntax of CCM $CCM ::= \phi \text{ ov } (\bot) \mid \phi \text{ ov } (0) \mid \phi \text{ ov } (l) \mid CCM_1 \| CCM_2$

Semantics of CCM

1. $\mathcal{I} \not\models \phi \text{ ov } (\bot)$ for any interpretation $\mathcal{I}$.
2. $\mathcal{I} \models \phi \text{ ov } (0)$ iff $\mathcal{I} \models \phi$.
3. $\mathcal{I} \models \phi \text{ ov } (l)$ iff one of the following cases holds:
   (a) $\mathcal{I} \Vdash l$ and there exists $\langle r_0, r_1, \ldots, r_n\rangle \in S_l^{\mathcal{I}}$, $n \in N_\omega$ such that $\sigma \downarrow \langle r_0, r_1, \ldots, r_h\rangle \models \phi$ for some $0 \le h \le n$, or
   (b) there exists r, $k \le r \preceq j$ such that $\mathcal{I}_1 = (\sigma, i, k, r) \Vdash l$ and there exists $\langle r_0, r_1, \ldots, r_n\rangle \in S_l^{\mathcal{I}_1}$, $n \in N_0$ and $\sigma \downarrow \langle r_0, r_1, \ldots, r_n\rangle \bullet \sigma_{(r_n+1..|\sigma|)} \models \phi$.
4. $\mathcal{I} \models CCM_1 \| CCM_2$ iff $\mathcal{I} \models (CCM_1; \mathsf{true}) \wedge CCM_2 \vee CCM_1 \wedge (CCM_2; \mathsf{true})$.

According to the semantics given above, it is readily to prove the following laws which are useful to prove the normal form of CCM and can also be used as reduction rules in the execution of programs.

L1.	ϕ ov $(\perp)$	$\equiv$ false		L6.	$\bigcirc\phi$ ov (n,l)	$\equiv$	$\bigcirc^n\varepsilon;(\phi$ ov $(l))$
L2.	ϕ ov (0)	$\equiv \phi$		L7.	$(w\wedge\phi)$ ov (l)	$\equiv$	$w\wedge(\phi$ ov $(l))$
L3.	ε ov (n)	$\equiv \bigcirc(\varepsilon$ ov $(n-1))$		L8.	ϕ ov (l_1+l_2)	$\equiv$	$(\phi$ ov $(l_1))\vee(\phi$ ov $(l_2))$
L4.	ε ov (n,l)	$\equiv \bigcirc(\varepsilon$ ov $(n-1,l))$		L9.	$(\phi_1\vee\phi_2)$ ov (l)	$\equiv$	$(\phi_1$ ov $(l))\vee(\phi_2$ ov $(l))$
L5.	$\bigcirc\phi$ ov (n)	$\equiv \bigcirc^n\varepsilon;\phi$					

4 Operational Semantics of CCM

Formal semantics of programming language is a prerequisite to reasoning properties of programs. Operational semantics defines the semantics of program in an operational manner with which we can understand how to execute a program and implement an interpreter for the programming language. In [23], the operational semantics of MSVL is discussed in detail. For ease of understanding of the operational semantics of CCM, we present briefly the basic idea of the work presented in [23].

To execute an MSVL program is to find a model for it. The execution of MSVL programs proceeds along with reduction. The reduction process of an interval temporal logic program can be divided into two phases: one for state reduction and the other for interval reduction. The state reduction is concerned with how to transform a program into its normal form. It consists of evaluation rules of arithmetic and boolean expressions, semantic equivalence rules and transition rules within a state. The interval reduction is concerned with a program executed over an interval. It includes a group of interval transition rules to transfer a program from one state to another. Therefore, evaluation rules of expressions, semantic equivalence rules, transition rules within a state and interval transition rules constitute the operational semantics of MSVL.

There are two types of configurations, one for expressions and the other for programs. A configuration for a program ϕ is a tuple $(\phi,\sigma_{i-1},s_i,i)$, where ϕ is a framed program, $\sigma_{i-1}=\langle s_0,\ldots,s_{i-1}\rangle(i>0)$ a model which records information of all states, s_i the current state at which ϕ is being executed and i a counter for counting the number of states in σ_{i-1}. Further, for $i=0$, let $\sigma_{-1}=\epsilon$ be an empty sequence. Thus, the initial configuration is $(\phi,\epsilon,s_0,0)$. When a program is terminating, it is reduced to true and the state is written as $\emptyset$. So the final configuration is $(\mathsf{true},\sigma,\emptyset,|\sigma|+1)$. For an arithmetic or boolean expression exp, the configuration is (exp,σ_{i-1},s_i,i). Evaluation rules map the configuration (e,σ_{i-1},s_i,i) to an element n in D which means the value of e under the current state s_i is calculated as n. Similarly, for an boolean expression b, configuration (b,σ_{i-1},s_i,i) is mapped to a truth value in B.

The normal form plays an essential role in the reduction of temporal logic programs. In the process of reducing a program into its normal form, we need a number of logical laws to allow convenient reasoning and transferring of programs. These logical laws are given as semantic equivalent rules.

Since adding framing operators to a program leads to the feature of non-monotonicity, we cannot capture the semantics of $\phi_1\wedge\phi_2$ by means of composing the semantics of ϕ_1 and ϕ_2. Further, in temporal logic programming, assignment statements are executed concurrently and may depend on one another such as the statement $(x = y+z) \wedge (z = y+x+2) \wedge (y = z+x)$. Transition rules within a state are used to deal with concurrent assignments within a state. With these rules, values of variables in an assignment statement like equations are evaluated in some order within a state and the minimal model is also captured.

In general, the reduction at each state may be done by several steps. Notation $\rightarrowtail$ is a binary relation over the set of configurations with a state. If the program ϕ in a configuration is of the form of $\bigcirc\phi'$ or empty which means that the reduction of the program needs to be moved to the next state or stopped, then interval transition rules are required to help the reduction progress. The transition relation between two configurations with different state is denoted by binary relation $\rightarrow$. Rule $\textsc{Tr}1$ $(\bigcirc\phi, \sigma_{i-1}, s_i, i) \rightarrow (\phi, \sigma_i, s_{i+1}, i+1)$ means that current state s_i is appended to model σ_{i-1}, ϕ requests to be executed at the next state s_{i+1}, and the number of states increases by one. Rule $\textsc{Tr}2$ means s_i is appended to σ_{i-1} and the final configuration $(\mathsf{true}, \sigma_i, \emptyset, i+1)$ is reached. For more details, refer to [23].

In this section, we extend the operational semantics of MSVL to include the operational semantics of CCM. To this end, we only need to give the semantic equivalence rules about sequence expressions and CCM which are used to transform CCM construct into its normal form. Firstly, we need to present Lemma 1 and Theorem 2.

Lemma 1. For any sequence expression $l \in L$, there exists a sequence expression $\overset{m}{\underset{i=1}{+}}(n_i, l_i)$ where $m, n_i \in N_0^+$ and $l_i \in L \cup \{0\}$ such that $l = \overset{m}{\underset{i=1}{+}}(n_i, l_i)$.

Lemma 1 tells us that for any sequence expression l, there exists a sum l' of sequence expressions which are headed by integers such that $l = l'$. This, in fact, is a prerequisite of transforming CCM construct into its normal form.

Theorem 2. Any CCM program can be transformed into normal form.

Theorem 2 means that any CCM program can be transformed into its normal form. Based on properties of sequence expressions and logical laws of CCM, the process of transformation of CCM into its normal form is also feasible. Based on these properties and logical laws, the operational semantics of CCM can be defined in the two boxes given below where $\mathsf{R1} - \mathsf{R14}$ are semantic equivalence rules for sequence expressions and $\mathsf{L1} - \mathsf{L11}$ semantic equivalence rules for CCM programs.

Example: Reduce the program $x = 1 \wedge \Box(\bigcirc x = x+1)\ \mathsf{ov}\ (2^{\circledast})$ by operational rules.

$$\begin{array}{ll}
\quad (x = 1 \wedge \Box(\bigcirc x = x+1)\ \mathsf{ov}\ (2^{\circledast}), \epsilon, s_0, 0) & \\
\rightarrowtail (x = 1 \wedge \bigcirc x = x+1 \wedge \bigcirc\Box(\bigcirc x = x+1)\ \mathsf{ov}\ (2^{\circledast}), \epsilon, s_0, 0) & \{\textsc{Alw}, \mathsf{L11}\} \\
\rightarrowtail (x = 1 \wedge \bigcirc x = 2 \wedge \bigcirc\Box(\bigcirc x = x+1)\ \mathsf{ov}\ (2^{\circledast}), \epsilon, s_0, 0) & \{\textsc{Sub-term}\} \\
\rightarrowtail (x = 1 \wedge \bigcirc(x = 2 \wedge \Box(\bigcirc x = x+1))\ \mathsf{ov}\ (2 + (2, 2^{\circledast})), \epsilon, s_0, 0) & \{\mathsf{R13}, \mathsf{L10}\} \\
\rightarrowtail (x = 1 \wedge \bigcirc(x = 2 \wedge \Box(\bigcirc x = x+1))\ \mathsf{ov}\ (2), \epsilon, s_0, 0) & \\
\quad \text{or } (x = 1 \wedge (x = 2 \wedge \Box(\bigcirc x = x+1))\ \mathsf{ov}\ (2, 2^{\circledast}), \epsilon, s_0, 0) & \{\mathsf{L8}\}
\end{array}$$

As we can see, the program is non-deterministic since the reduction generates two different configurations. The first one is deterministic and the second one is non-deterministic. This result is determined by the non-deterministic semantics of $\circledast$. We take the first one as an example.

R1.	$\bot, l = l, \bot = \bot$
R2.	$l_1 + l_2 = l_2 + l_1$
R3.	$0, l = l, 0 = l$
R4.	$l_1, (l_2 + l_3) = l_1, l_2 + l_1, l_3$
R5.	$l + l = l$
R6.	$(l_1 + l_2), l_3 = l_1, l_3 + l_2, l_3$
R7.	$l, l^{\textcircled{r}} = l^{\textcircled{r}}, l$
R8.	$l_1, (l_2, l_1)^{\textcircled{r}} = (l_1, l_2)^{\textcircled{r}}, l_1$
R9.	$l^{\textcircled{r}}, l^{\textcircled{r}} = l, l^{\textcircled{r}}$
R10.	$l_1, (l_2, l_3) = (l_1, l_2), l_3$
R11.	$(l^{\textcircled{r}})^{\textcircled{r}} = l^{\textcircled{r}}$
R12.	$l_1 + (l_2 + l_3) = (l_1 + l_2) + l_3$
R13.	$l^{\textcircled{r}} = l + l, l^{\textcircled{r}}$
R14.	$\dfrac{l_1 = l_2}{l = l[l_2/l_1]}$

L1.	ϕ ov $(\bot)$	$\equiv$ false
L2.	ϕ ov (0)	$\equiv \phi$
L3.	ε ov (n)	$\equiv \bigcirc(\varepsilon$ ov $(n-1))$
L4.	ε ov (n, l)	$\equiv \bigcirc(\varepsilon$ ov $(n-1, l))$
L5.	$\bigcirc\phi$ ov (n)	$\equiv \bigcirc^n \varepsilon; \phi$
L6.	$\bigcirc\phi$ ov (n, l)	$\equiv \bigcirc^n \varepsilon; (\phi$ ov $(l))$
L7.	$(w \wedge \phi)$ ov (l)	$\equiv w \wedge (\phi$ ov $(l))$
L8.	ϕ ov $(l_1 + l_2)$	$\equiv (\phi$ ov $(l_1)) \vee (\phi$ ov $(l_2))$
L9.	$(\phi_1 \vee \phi_2)$ ov (l)	$\equiv (\phi_1$ ov $(l)) \vee (\phi_2$ ov $(l))$
L10.	$\dfrac{l_1 = l_2}{\text{prog} \equiv \text{prog}[l_2/l_1]}$	
L11.	$\dfrac{q \equiv q'}{\text{prog} \equiv \text{prog}[q'/q]}$	

$$
\begin{array}{lr}
\quad (x = 1 \wedge \bigcirc(x = 2 \wedge \Box(\bigcirc x = x + 1)) \text{ ov } (2), \epsilon, s_0, 0) & \\
\rightarrowtail (x = 1 \wedge \bigcirc^2 \varepsilon; (x = 2 \wedge \Box(\bigcirc x = x + 1)), \epsilon, s_0, 0) & \{\text{L5}\} \\
\rightarrowtail (\bigcirc^2 \varepsilon; (x = 2 \wedge \Box(\bigcirc x = x + 1)), \epsilon, s_0[(1, \neg p_x)/x], 0) & \{\textsc{Min2}\} \\
\rightarrow (\bigcirc \varepsilon; (x = 2 \wedge \Box(\bigcirc x = x + 1)), \langle s_0[(1, \neg p_x)/x] \rangle, s_1, 1) & \{\textsc{Tr1}\} \\
\rightarrowtail (\bigcirc(x = 2 \wedge \Box(\bigcirc x = x + 1)), \langle s_0[(1, \neg p_x)/x] \rangle, s_1, 1) & \{\textsc{Chop}\} \\
\rightarrow (x = 2 \wedge \Box(\bigcirc x = x + 1), \langle s_0[(1, \neg p_x)/x], s_1 \rangle, s_2, 2) & \{\textsc{Tr1}\} \\
\rightarrowtail (x = 2 \wedge \bigcirc x = x + 1 \wedge \bigcirc\Box(\bigcirc x = x + 1), \langle s_0[(1, \neg p_x)/x], s_1 \rangle, s_2, 2) & \{\textsc{Alw}, \text{L11}\} \\
\rightarrowtail (x = 2 \wedge \bigcirc(x = 3 \wedge \Box(\bigcirc x = x + 1)), \langle s_0[(1, \neg p_x)/x], s_1 \rangle, s_2, 2) & \{\textsc{Sub-term}\} \\
\rightarrowtail (\bigcirc(x = 3 \wedge \Box(\bigcirc x = x + 1)), \langle s_0[(1, \neg p_x)/x], s_1 \rangle, s_2[(2, \neg p_x)/x], 2) & \{\textsc{Min2}\} \\
\rightarrow (x = 3 \wedge \Box(\bigcirc x = x + 1), \langle s_0[(1, \neg p_x)/x], s_1, s_2[(2, \neg p_x)/x] \rangle, s_3, 3) & \{\textsc{Tr1}\}
\end{array}
$$

The reduction process given above generates a prefix model which contains only three states. Since the program is a non-terminable program, the reduction process will never terminate. The reduction rules ALW, CHOP, SUB-TERM, MIN2 and TR1 can be found in the operational semantics of MSVL [23].

5 Implementation of CCM in MSVL Interpreter

The implementation of CCM is similar to that of MSVL statements which has been given in [6]. Basically, the implementation strategy for MSVL is based on the normal form. That is, to execute a program is to reduce it into a logically equivalent conjunction: *Present* $\wedge$ *Remains* where *Present* consists of assignments (by $=$ or $\Leftarrow$) to program variables, output of program variables, true and more or empty. more and empty are used to indicate whether or not the interval is terminated. Formally,

$$Present = \bigwedge_{i=1}^{m} present_i \qquad present_i ::= \ x = e \mid x \Leftarrow e \mid \text{true} \mid \text{more} \mid \text{empty}$$

The *Remains* is what is executed in the subsequent state, if any. It is in a *reduced form* if either it is true or it consists of conjuncts leading with only the next operator. Formally,

$$Remains = \bigwedge_{i=1}^{n} \bigcirc w_i$$

where w_i is a MSVL program. When preparing the execution of the next state, a function *next_w* is used to remove these next operators from the conjuncts contained in *Remains* and what is really executed at the next state is the formula *Next*

$$Next = \bigwedge_{i=1}^{n} w_i \qquad\qquad w_i = next\_w(\bigcirc w_i)$$

To manage the reductions, we need to define an important flag, *done*, which indicates whether or not the interval is terminated. At the beginning of the reduction in each state, *done* is set to *nil*, and at the ending of each state, the interpreter sets its value to either true or false, depending on if the empty or more conjunct has been encountered. If the program fails to specify the status of the interval, the interpreter cannot set the *done* which will remain equal to *nil*. Then an error will be detected and indicated.

The execution of an MSVL program proceeds through a number of states. The execution at a state is composed of several passes of reductions. After the last pass, the program is reduced to the form of *Present* $\wedge$ *Remains*. *Present* is dissolved during the reduction. Its effect is reflected in updating and displaying the values of variables, setting the *done* flag, etc. What remains after the last pass of reduction is *Remains* which will be executed at the next state if the interval over which the formula is executed does not end. For more details of the interpreter of MSVL, refer to Chapter 9 in [6].

There are two main operators in CCM, one is *parallel* ($\|$), the other is *over* (ov). Since *parallel* has been implemented in [6], here we introduce it briefly for the sake of being self-contained and then we give the implementation of *over*.

Reduction of *parallel*. The interpreter handles a statement of the form $\phi_1 \| \phi_2$ by transforming the statement to the internal construct $parallel(\phi_1, \phi_2, done(nil))$ which is immediately re-reduced. Here $done(nil)$ is a flag initialized to nil. It serves as a local $done$ flag for the interval over which the statement ϕ_1 is executed.

$parallel(\phi_1, \phi_2, done(D1))$ construct is executed by first saving the value of the old $done$ flag to OLD and setting it to $D1$. The statement ϕ_1 is then reduced in the context to a new statement ϕ_1'. Afterwards the current $done$ flag is saved to $D2$, and the old $done$ flag value, OLD, is restored. The statement ϕ_2 is then reduced in the context to a new statement ϕ_2'. If ϕ_1' or ϕ_2' is not fully reduced, then the parallel statement can be written as $parallel(\phi_1', \phi_2', done(D2))$. This is returned as the result of the reduction. On the other hand, if ϕ_1' and ϕ_2' are both fully reduced, then the overall parallel is transformed to the following conditional statement and then immediately re-reduced:

```
if $(done(OLD) = done(\mathtt{true}))$ then $(\phi1, done(D2))$
        else if $(done(D2) = done(\mathtt{true}))$ then $\phi2$
            else $\bigcirc\, parallel(\phi1, \phi2, done(nil))$
```

Here $\phi1 = next\_w(\phi_1')$ and $\phi2 = next\_w(\phi_2')$, and $(\phi1, done(D2))$ means to set $done$ flag to $D2$ and then to reduce $\phi1$ in the context. For more details, refer to [6].

Reduction of *over*. The interpreter handles ϕ ov (l') $(l' \in L')$ by first separating an integer from sequence expression l'. By Lemma 1, any sequence expression $l \in L$ can be reduced to an union of several sequence expressions each headed by an integer. This process is described in algorithm **IntSep**(ϕ, S_e) where ϕ is an MSVL program and S_e is a sequence expression.

```
IntSep($\phi$, $S_e$)
begin
  switch $S_e$ on :
    case ($\bot$) :            false;
    case (0) :                 $\phi$;
    case ($n$) :               $over(\phi, n, 0)$;
    case ($n, l$) :            $over(\phi, n, l)$;
    case (($l_1 + l_2$), $l$) : or (IntSep($\phi$, ($l_1, l$)), IntSep($\phi$, ($l_2, l$)));
    case ($l_1^{\circledast}, l$) :  or (IntSep($\phi$, ($l_1, l$)), IntSep($\phi$, ($l_1, l_1^{\circledast}, l$)));
    case ($l^{\circledast}$) :       or (IntSep($\phi$, ($l$)), IntSep($\phi$, ($l, l^{\circledast}$)));
    case ($l_1 + l_2$) :       or (IntSep($\phi$, ($l_1$)), IntSep($\phi$, ($l_2$))).
end
```

The interpreter processes the construct $over(\phi, n, R)$ by first allocating for ϕ a *done* flag initiated to nil and transforming it into the internal construct $\mathsf{ov}(R, \phi, n, done)$. The statement ϕ is then reduced to ϕ'. If it is not fully reduced, $\mathsf{ov}(R, \phi', n, done)$ is returned as the result which will be continued to reduce until ϕ is in its reduced form. Otherwise, ϕ is transformed into the form of $\bigcirc\phi_1$, then the overall ov statement is transformed into the following statement:

$\texttt{if } (done = true) \texttt{ then } \bigcirc (\bigcirc^{n-1}\varepsilon; (\varepsilon \texttt{ ov } (R))) \texttt{ else } \bigcirc (\bigcirc^{n-1}\varepsilon; (\phi_1 \texttt{ ov } (R)))$

The execution of " or $(program1, program2)$" is easy to understand. It is the task of the interpreter to decide which program to execute.

6 Case Study: A Word Processor

A word processor generally does the following tasks: (1) Read characters and collect them as words; (2) Collect words to fill a line; (3) Hyphenate, if necessary; (4) Collect enough lines to make a page and print the page. To give a parallel algorithm of the word processor, the first thing we need to do is to analyze the parallelism among the tasks. From the above, (2) and (3) tasks must be sequential since when the collection of a line is over, we can determine whether or not hyphenation is needed and the last word will be split. Further, (1), (2,3) and (4) tasks can be processed in parallel. So the function of a word processor can be divided into three sub-functions and each function can be implemented by a process. Three processes are hosted over three cores respectively and executed in parallel. With MSVL, these three processes can be specified as $Word$, $Line$ and $Page$. As a result, the word processor can be specified briefly as the following CCM program:

$WP \overset{\text{def}}{=} \mathsf{frame}(i, B, n, W, WL, over, flag, t, q, L, lp) \texttt{ and } (Input; (Word \| Line \| Page))$

$Line \overset{\text{def}}{=} Lflag \| Lctr$

Word processor first receives the characters input by users and ended with symbol '$\#'$, preserves them in an array B, then analyzes the characters and collects them into words, collects words to fill a line, and prints a page if the page is full. Constants and variables and procedures needed in the program are given as follows:

l : an integer constant, the width of line;

w : an integer constant, the average width of characters;

m : an integer constant, specifies the number of lines in a page;

i : an integer variable, records the total number of characters;

n : an integer variable, records the total number of collected words;
t : an integer variable, controls the subscript of L;
q : an integer variable, controls the subscript of W and WL when updating L.
$over$: a boolean variable, indicates whether $Lflag$ terminates;
$flag$: a boolean variable, $flag = 1$ indicates the starting of $Lctr$;
lp : an integer variable, controls the begin and end line of Print;
B : a char array, preserves the input characters;
L : an integer array, the i-th line is composed of word from $W[L[i-1]+1]$ to $W[L[i]]$;
W : a string array, preserves the collected words;
WL : a float-point array, records the length of the words in W;
String(A :array, x :int, y :int) : a procedure transforms an array into a string;
Length(s :string) : a procedure returning the length of a string;
Need(s:string): a procedure decides whether a word s can be separated or not.
Hyphen(s:string): a procedure returns the position of the separation of s.
Cut(s:string,h:int): a procedure divides s at the h-th character, the first part is placed in the "first" composition of structured variable and the second part in the "second" composition.
Print(j:int): print a page contains from the $j * m$-th line to the $(j+1) * m - 1$-th line.

We use the procedures to assign some variables that have not been assigned values. The definitions of components of the word processor are given as follows.

```
$Input \overset{\text{def}}{=}$ $i \Leftarrow 0$ and $input(B[i])$;
        while( not $(B[i] =' \#'))\{i :=^{\dagger} i+1$ and $input(B[i+1])\}$

$Word \overset{\text{def}}{=}$ $n \Leftarrow 0$ and $j = 0$ and
        while($j < i$)
        { $x = 0$ and
        while( not $(B[j] =' \ '$ or $','$ or $'.'$ or $';'$ or $'''$ or $'"'$ or $'/'))$
        $\{A[x] = B[j]$ and $x := x+1$ and $j := j+1\}$;
        $A[x] = B[j]$ and $W[n] \Leftarrow$ String$(A, 0, x)$
         and $WL[n] \Leftarrow$ Length$(W[n])$
         and $n :=^{\dagger} n+1$ and $j := j+1\}$
$Lflag \overset{\text{def}}{=}$ $(over \Leftarrow 0$ and next always$(flag \Leftarrow 1); over \Leftarrow 1)$ ov $((\lceil \frac{l}{w} \rceil)^{ⓡ})$
$Lctr \overset{\text{def}}{=}$ $t \Leftarrow 0$ and $q \Leftarrow 0$ and
        while( not $(over = 1$ and $flag = 0))$
        { await$(flag = 1)$;
         $flag \ \sigma\!\!=^{\dagger} 0$ and $S = 0$ and
         while$(S < l)\{S := S + WL[q]$ and $q :=^{\dagger} q+1\}$;
         $L[t] \Leftarrow q-1$ and $t :=^{\dagger} t+1$;
         if $(S > l)$
         then if (Need$(W[L[t-1]]))$
              then $(h =$ Hyphen$(W[L[t-1]])$ and
                   $W[L[t-1]] :=^{\dagger}$ Cut$(W[L[t-1]], h).first \bullet$ "$-$" and
                   $WL[L[t-1]] :=^{\dagger} h+1$ and
                   $W[L[t-1]+1] :=^{\dagger}$ Cut$(W[L[t-1]], h).second \bullet W[L[t-1]+1]$ and
                   $WL[L[t-1]+1] :=^{\dagger} WL[L[t-1]+1] + WL[L[t-1]] - h)\}$
$Page \overset{\text{def}}{=}$ $(lp \Leftarrow 0$ and
        next always (if $(t \geq (lp+1) * m)$ then Print$(lp)$ and $lp \ \sigma\!\!=^{\dagger} lp+1))$ ov $((\lceil \frac{lm}{w} \rceil)^{ⓡ})$
```

7 Related Work

Since 1970's, a number of parallel or concurrent programming languages with one single processor based on interleaving or true concurrency model have been formalized [1,2,3,4,8,9,10,11,12,13,14,15,17,19,20,21,22]. For instance, with process algebra community, CSP [2] and CCS [3,4] are the typical concurrent languages which can be used to specify and verify concurrent systems. Nevertheless, they are not executable. Temporal logics is a useful formalism for specifying and verifying correctness of concurrent programs. Many programming languages based on temporal logics have been proposed so as to specify and reason about programs in the same logic framework. Cactus [19] is based on branching-time temporal logic. XYZ/E [20], THLP [21], Chronolog [22], Templog [1] and Temporal Prolog [15] are based on linear-time temporal logic. Tempura [17] and Tokio [14] are based on interval temporal logic. TLA is also a specification language based on temporal logic of actions [8]. Moreover, many other concurrent languages based on Petri nets were also proposed in the past 40 years such as Colored-PN and Timed-PN [9,10,11,12,13]. Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution. Petri nets also have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis. However, the purpose of these languages is for specifying and verifying rather than for programming of concurrent systems. Furthermore, most of languages given above are concerned with one single processor and used for dealing with concurrent processes under interleaving or true concurrency model. It is possible to work out various kinds of multiprocessor programming languages based on the above mentioned ones but there is no a straightforward way to do the job.

Multiprocessor (or multi core or many core) programming has been popular since the turn of the century. As clock speed cannot be increased without overheating, manufactures have to turn to multi-core architectures, in which multiple processors (cores) cooperate and communicate directly through shared hardware caches. The idealized computation model is based on concurrent threads operating a set of shared objects. The sequence of the thread operations on the objects is called the concurrent program. Programming languages such as Java, C#, or C++ threads are using this model. Surprisingly, the shared objects are easy to specify but not easy to implement by any concurrent algorithm. The correctness of multiprocessor programs is more complex than that of their sequential counterparts. As a result, multiprocessors or multi-core (many core) programming is a challenge for programmers to develop correct programs. Thus, fundamental multiprocessors programming technologies need to be explored and a different set of supporting tools such as model checkers and theorem provers need to be developed.

In contrast with the above formalisms, MSVL is of distinct advantages because it is a temporal logic programming language with three modes: modeling, simulation and verification. In particular, CCM construct $\phi_1 \texttt{ ov } (l_1) \| \ldots \| \phi_m \texttt{ ov } (l_m)$ is better suited to multi-core programming. It enables us not only to specify the shared objects but also to program and to verify the system in a convenient way. In fact, it provides us both a semantic abstract model based on shared variables and a programming language for multi-core programming.

8 Conclusions

We presented a semantic model CCM for many-core or many-task computation systems. To realize the model with MSVL paradigm, CCM programs are defined and implemented in MSVL. The implementation strategy and the reduction rules are also presented. As an example of many-core systems, a word processor is formalized and implemented in MSVL using CCM programs. This enables us to handle many-core (or many-task) systems in a formal and flexible way.

As you can see, CCM programs can be used to manage many-core computation naturally. However, we need to do some further case studies for more complex many-core computation on a many-core platform. This is a challenge to us in the near future. Further, we need to apply CCM programs to specify and verify many-core systems. To do so, we also need to develop a new interpreter on a many-core platform to manage the specification and verification of many-core systems.

References

1. Abadi, M., Manna, Z.: Temporal Logic Programming. Journal of Symbolic Computation 8, 277–295 (1989)
2. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, London (1985)
3. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
4. Milner, R.: Communication and Concurrency. Prentice Hall, London (1989)
5. Duan, Z., Koutny, M., Holt, C.: Projection in Temporal Logic Programming. In: Pfenning, F. (ed.) LPAR 1994. LNCS (LNAI), vol. 822, pp. 333–334. Springer, Heidelberg (1994)
6. Duan, Z.: Temporal Logic and Temporal Logic Programming. Science Press, Beijing (2005)
7. Duan, Z., Yang, X., Koutny, M.: Frammed Temporal Logic Programming. Science of Computer Programming 70, 31–61 (2008)
8. Lamport, L.: The temporal logic of actions. ACM Trans. Program. Lang. Syst. 16, 872–923 (1994)
9. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. International Journal on Software Tools for Technology Transfer 9, 213–254 (2007)
10. Jensen, K.: Coloured Petri Nets: A High Level Language for System Design and Analysis. In: Rozenberg, G. (ed.) APN 1990. LNCS, vol. 483, pp. 342–416. Springer, Heidelberg (1991)
11. Koutney, M., Pietkiewicz-Koutney, M.: Synthesis of Petri Nets with Localities. Sci. Ann. Comp. Sci. 19, 1–23 (2009)
12. Koutny, M.: A compositional model of time petri nets. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 303–322. Springer, Heidelberg (2000)
13. Billington, J., Christensen, S., Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The petri net markup language: Concepts, technology, and tools. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 483–505. Springer, Heidelberg (2003)
14. Fujita, M., Kono, S., Tanaka, H., Moto-oka, T.: Tokio: Logic Programming Language Based on Temporal Logic and its Compilation to PROLOG. In: Shapiro, E. (ed.) ICLP 1986. LNCS, vol. 225, pp. 695–709. Springer, Heidelberg (1986)
15. Gabbay, D.M.: Modal and Temporal Logic Programming. Temporal Logics and their Applications, pp. 197–237. Academic Press, London (1987)

16. Keller, R.M.: Formal Verification of Parallel Programs. Communications of the ACM 19(7), 371–384 (1976)
17. Moszkowski, B.C.: Executing Temporal Logic Programs. Cambridge University Press, Cambridge (1986)
18. Owicki, S., Gries, D.: Verifying Properties of Parallel Programs: An Axiomatic Approach. Communications of the ACM 19(5), 279–285 (1976)
19. Rondogiannis, R., Gergatsoulis, M., Panayiotopoulos, T.: Cactus: A Branching Time Logic Programming Languages. In: Nonnengart, A., Kruse, R., Ohlbach, H.J., Gabbay, D.M. (eds.) FAPR 1997 and ECSQARU 1997. LNCS (LNAI), vol. 1244, pp. 511–524. Springer, Heidelberg (1997)
20. Tang, C.S.: Toward a Unified Logical Basis for Programming Languages. In: Proceedings of IFIP Congress, pp. 425–429 (1983)
21. Wadge, W.W.: Tense Logic Programming: A Respectable Alternative. In: Proceedings of the 1988 International Symposium on Lucid and Intensional Programming, pp. 26–32 (1988)
22. Orgun, M.A., Wadge, W.W.: A Theory and Practice of Temporal Logic Programming. Intensional Logics for Programming, pp. 23–50. Oxford University Press, Oxford (1992)
23. Yang, X., Duan, Z.: Operational Semantics of Framed Tempura. Journal of Logic and Algebraic Programming 78, 22–51 (2008)

On Unique Games with Negative Weights

Peng Cui[1], Tian Liu[2], and Ke Xu[3,⋆]

[1] Key Laboratory of Data Engineering and Knowledge Engineering, MOE, School of Information Resource Management, Renmin University of China, Beijing 100872, P.R. China
cuipeng@ruc.edu.cn

[2] Key Laboratory of High Confidence Software Technologies, Ministry of Education, Institute of Software, School of Electronic Engineering and Computer Science, Peking University, Beijing 100871, P.R. China
lt@pku.edu.cn

[3] National Lab of Software Development Environment, Beihang University, Beijing 100083, P.R. China
kexu@nlsde.buaa.edu.cn

Abstract. In this paper, the authors define Generalized Unique Game Problem (GUGP), where weights of the edges are allowed to be negative. Focuses are made on two special types of GUGP, GUGP-NWA, where the weights of all edges are negative, and GUGP-PWT(ρ), where the total weight of all edges are positive and the negative/positive ratio is at most ρ. The authors investigate the counterpart of the Unique Game Conjecture on GUGP-PWT(ρ). The authors prove Unique Game Conjecture holds true on GUGP-PWT(1) by reducing the parallel repetition of Max 3-Cut Problem to GUGP-PWT(1), and Unique Game Conjecture holds true on GUGP-PWT(1/2) if the 2-to-1 Conjecture holds true. The authors pose an open problem whether Unique Game Conjecture holds true on GUGP-PWT(ρ) with $0 < \rho < 1$.

1 Introduction

The Unique Game Conjecture is put forward by Khot on STOC 2002 as a powerful tool to prove lower bound of inapproximabilty for combinatorial optimization problems[1]. It has been shown by researchers a positive resolution of this conjecture would imply improved even best possible hardness results for many famous problems, to name a few, Max Cut, Vertex Cover, Multicut, Min 2CNF Deletion, making an important challenge to prove or refute the conjecture.

Some variations of the Unique Game Conjecture have been mentioned. Rao proves a strong parallel repetition theorem which shows Weak Unique Game Conjecture is equivalent to the Unique Game Conjecture[2]. Khot et al. show that Unique Game Conjecture on Max 2LIN(q) is equivalent to the Unique Game Conjecture[3]. Khot pose

⋆ This research was partially supported by the National 973 Program of China (Grant No. 2010CB328103) and the National Natural Science Foundation of China (Grant Nos. 60725207 and 60973033), and by the Research Project of Renmin University of China (Grant No. 2009060064).

the d-to-1 Conjectures in his original paper for $d \geq 2$[1]. O'Donell et al. show a tight hardness for approximating satisfiable constraint satisfaction problem on 3 Boolean variables assuming the d-to-1 Conjecture for any fixed d[4]. Dinur et al. use the 2-to-2 Conjecture and the $\ltimes$ conjecture to derive the hardness results of Approximate Coloring Problem, and prove that the (exact) 2-to-1 Conjecture implies their 2-to-2 Conjecture[5]. Guruswami et al. use the (exact) 2-to-1 Conjecture to derive the hardness result of Maximum k-Colorable Subgraph Problem[6]. It is unknown whether the Unique Game Conjecture implies any of the d-to-1 Conjectures, or vice versa.

Almost concurrently to [1], Khot defines the smooothness property of 2-Prover 1-Round Game (2P1R), a weaker analogue of the property of Unique Game Problem[7]. He constructs instances of 2P1R that are η-smooth for arbitrarily small η to prove hardness of Coloring 3-Uniform Hypergraphs. 2P1R with certain smooothness properties have been used to prove some other hardness results[8,9,10].

In this paper, the authors relax Unique Game Problem (UGP) by defining Generalized Unique Game Problem (GUGP), where weights of the edges are allowed to be negative. Focuses are made on two special types of GUGP, GUGP-NWA, where the weights of all edges are negative, and GUGP-PWT(ρ), where the total weight of all edges are positive and the negative/positive ratio is at most ρ. Max GUGP-NWA can be restated as 2P1R where each relation is a complement of a permutation. GUGP-PWT(ρ) over $1 \geq \rho \geq 0$ makes a possible phase transition from a hard $(1-\zeta, \delta)$-gap problem of 2P1R to UGP.

The authors investigate the counterpart of the Unique Game Conjecture on GUGP-PWT(ρ). The authors prove Unique Game Conjecture holds true on GUGP-PWT(1) by reducing the parallel repetition of Max 3-Cut Problem to GUGP-PWT(1), and Unique Game Conjecture holds true on GUGP-PWT(1/2) if the 2-to-1 Conjecture holds true. It is shown the $(1-\zeta, \delta)$-gap hardness of problems in GUGP-PWT(ρ) possesses the compactness property when $\rho \to 0$. The authors pose an open problem whether Unique Game Conjecture holds true on GUGP-PWT(ρ) with $0 < \rho < 1$.

Section 2 demonstrates some definitions and conjectures. The authors show the main results for GUGP-NWA in Section 3 and for GUGP-PWT(ρ) in Section 4. Section 5 is some discussions.

2 Preliminaries

In 2-Prover 1-Round Game Problem (2P1R), we are given a bipartite graph $G = (V, W; E)$, with each edge e having a weight $w_e \in \mathbb{Q}^+$. We are also given two sets of labels, k_1 and k_2, which we identify with $[k_1] = \{1, \cdots, k_1\}$ and $[k_2] = \{1, \cdots, k_2\}$. Each edge $e = (u, v)$ in the graph is equipped with a relation $R_e \subseteq [k_1] \times [k_2]$. The solution of the problem is a labeling $f_1 : V \to [k_1]$ and $f_2 : W \to [k_2]$ which assigns a label to each vertex of G. An edge $e = (u, v)$ is said to be satisfied under f_1 and f_2 if $(f_1(u), f_2(v)) \in R_e$, else is said to be unsatisfied. The object of the problem is to find a labeling maximizing the total weight of the satisfied edges. The value of the instance, $Val(G)$, is defined as the maximum total weight of the satisfied edges divided by the total weight of all edges.

An instance of 2P1R has the projection property if all relations which the edges are equipped with are projections. We say an instance of 2P1R with the projection property is η-smooth (or has η-smoothness property) if for every $u \in V$ and two distinct labels $i, j \in [k_1]$, we have $Pr_v[\sigma_{(u,v)}(i) = \sigma_{(u,v)}(j)] \leq \eta$, where v is a randomly chosen neighbor of u, and $\sigma_{(u,v)}$ is the projection which the edge (u, v) is equipped with.

The following proposition describes the hardness of 2P1R with the smoothness property:

Proposition 1. ([10] Theorem 3.5) *For every $\eta, \delta > 0$, there are $k_1 = k_1(\eta, \delta)$ and $k_2 = k_2(\eta, \delta)$ such that given an η-smooth instance G of 2P1R with the label sets $[k_1]$ and $[k_2]$ it is NP-hard to distinguish whether $Val(G) = 1$ or $Val(G) < \delta$.*

Unique Game Problem (UGP) can be viewed as 2P1R with 0-smoothness property. In UGP, we are given an graph $G = (V, E)$, a weight function $w_e \in \mathbb{Q}^+$ for $e \in E$, and a set of labels, $[k]$. Each edge $e = (u, v)$ in the graph is equipped with a permutation $\pi_e : [k] \to [k]$. The solution of the problem is a labeling $f : V \to [k]$ which assigns a label to each vertex of G. An edge $e = (u, v)$ is said to be satisfied under f if $\pi_e(f(u)) = f(v)$, else is said to be unsatisfied. Note that we allow G is a graph with parallel edges, i.e., there exist more than one edge between two vertices.

It is possible to define two optimization problems in this situation. In Max UGP, the value of the instance is defined as the maximum total weight of the satisfied edges divided by the total weight of all edges. In Min UGP, the value of the instance is defined as the minimum total weight of the unsatisfied edges divided by the total weight of all edges.

Khot initiates much of the interest in the following conjecture by showing that many hardness results stem from it. It basically states that it is NP-hard to distinguish whether many or only few edges are satisfied.

Conjecture 1. ([1] Unique Game Conjecture in Max UGP Form) *For every $\zeta, \delta > 0$, there is a $k = k(\zeta, \delta)$ such that given an instance G of Max UGP with k labels it is NP-hard to distinguish whether $Val(G) > 1 - \zeta$ or $Val(G) < \delta$.*

The conjecture can be restated in Min UGP form, and the two conjectures are equivalent.

Conjecture 2. (Unique Game Conjecture in Min UGP Form) *For every $\zeta, \delta > 0$, there is a $k = k(\zeta, \delta)$ such that given an instance G of Min UGP with k labels it is NP-hard to distinguish whether $Val(G) < \zeta$ or $Val(G) > 1 - \delta$.*

2-to-1 Game and 2-to-2 Game are two special types of 2P1R. In 2-to-1 Game, we are given a bipartite graph $G = (V, W; E)$, with each edge e having a weight $w_e \in \mathbb{Q}^+$. We are also given two sets of labels, $[2k]$ for V and $[k]$ for W. Each edge $e = (u, v)$ in the graph is equipped with a 2-to-1 projection. A projection $\sigma : [2k] \to [k]$ is said to be a 2-to-1 projection if for each element $j \in [k]$ we have $|\sigma^{-1}(j)| = 2$. The value of the instance of 2-to-1 Game, $Val(G)$, is defined as the minimum total weight of the unsatisfied edges divided by the total weight of all edges.

In 2-to-2 Game, we are given an graph $G = (V, E)$, a weight function $w_e \in \mathbb{Q}^+$ for $e \in E$, and a set of labels, $[k]$. Each edge $e = (u, v)$ in the graph is equipped with a 2-to-2 relation. A relation $R \subseteq [2k] \times [2k]$ is said to be a 2-to-2 relation if there are two permutations $\pi_u, \pi_v : [2k] \to [2k]$ such that $(i, j) \in R$ iff $(\pi_u(i), \pi_v(j)) \in T$ where

$$T := \bigcup_{l=1}^{k} \{(2l-1, 2l-1), (2l-1, 2l), (2l, 2l-1), (2l, 2l)\}.$$

The value of the instance of 2-to-2 Game, $Val(G)$, is defined as the minimum total weight of the unsatisfied edges divided by the total weight of all edges.

The authors list the (exact) 2-to-1 Conjecture and the 2-to-2 Conjecture in their minimization forms. The latter is somewhat different from that in [5]. It can be proved that the 2-to-1 Conjecture implies our 2-to-2 Conjecture along the line of [5].

Conjecture 3. (2-to-1 Conjecture) *For every $\delta > 0$, there is a $k = k(\delta)$ such that given an instance G of 2-to-1 Game with the label sets $[2k]$ and $[k]$ it is NP-hard to distinguish whether $Val(G) = 0$ or $Val(G) > 1 - \delta$.*

Conjecture 4. (2-to-2 Conjecture) *For every $\delta > 0$, there is a $k = k(\delta)$ such that given an instance of 2-to-2 Game with the label set $[2k]$ it is NP-hard to distinguish whether $Val(G) = 0$ or $Val(G) > 1 - \delta$.*

In this paper, the authors define Generalized Unique Game Problem (GUGP), where weights of the edges are allowed to be negative. In GUGP, we are given an graph $G = (V, E)$ possibly having parallel edges, weight function $w_e \in \mathbb{Q}$ for $e \in E$, and the set of labels, $[k]$. Each edge $e = (u, v)$ in the graph is equipped with a permutation $\pi_e : [k] \to [k]$.

Note that w_e could be positive or negative. We assume there is no edge with zero weight for sake of clearance. Let W_G^+ be the total of the positive weights of all edges, W_G^- be the total of the negative weights of all edges, and $\Sigma_G = W_G^+ + W_G^-$ be the total weight of all edges. We call $r_G = |W_G^-|/W_G^+$ the negative/positive ratio of the instance.

The solution of GUGP is a labeling $f : V \to [k]$ which assigns a label to each vertex of G. It is also possible to define two optimization problems. In Max GUGP, the goal of is to maximize the total weight of the satisfied edges. In Min GUGP, the goal is to minimize the total weight of the unsatisfied edges. GUGP-NWA and GUGP-PWT are two special types of GUGP. In GUGP-NWA, the weight of all edges are negative. In GUGP-PWT, the total weight of all edges is positive.

In Max GUGP-NWA, we seek to minimize the total weight of the unsatisfied edges, i.e. to maximize the *absolute value* of the total weight of the unsatisfied edges. The value of Max GUGP-NWA is defined as the maximum absolute value of the total weight of the unsatisfied edges divided by $|W_G^-|$.

In Min GUGP-NWA, we seek to maximize the total weight of the satisfied edges, i.e. to minimize the *absolute value* of the total weight of the satisfied edges. The value of Min GUGP-NWA is defined as the minimum absolute value of the total weight of the satisfied edges divided by $|W_G^-|$.

In Max GUGP-PWT, we seek to maximize the total weight of the satisfied edges. The value of Max GUGP-PWT is defined as the maximum total weight of the satisfied edges divided by Σ_G. In an instance G of Max GUGP-PWT, let $W_G(f)$ be the total weight of the satisfied edges under labeling f, let the optimal labeling be f^*. The value of the instance is $Val(G) = W_G(f^*)/\Sigma_G$.

In Min GUGP-PWT, we seek to minimize the total weight of the unsatisfied edges. The value of Min GUGP-PWT is defined as the minimum total weight of the unsatisfied edges divided by Σ_G. In an instance G of Min GUGP-PWT, let $W_G(f)$ be the total weight of the unsatisfied edges under labeling f, let the optimal labeling be f^*. The value of the instance is $Val(G) = W_G(f^*)/\Sigma_G$.

We remaind the reader that the value of Max GUGP-PWT and Min GUGP-PWT could be negative or more than 1, although we still use the customary words "Completeness" and "Soundness" in our proofs of Theorem 3 and Theorem 4.

We define Max/Min GUGP-PWT(ρ) as the subproblem of Max/Min GUGP-PWT where the negative/positive ratio of the instances is upper bounded by ρ, where ρ is a constant independent from k. Since the negative/positive ratio is always less than 1, we set the range of ρ to be $0 \leq \rho \leq 1$. Note that Max/Min GUGP-PWT(0) is just Max/Min UGP.

We give the two equivalent counterparts of the Unique Game Conjecture on Max GUGP-PWT(ρ) and Min GUGP-PWT(ρ)) as follows:

Conjecture 5. (Unique Game Conjecture on Max GUGP-PWT(ρ)) *For every $\zeta, \delta > 0$, there is a $k = k(\zeta, \delta)$ such that given an instance of Max GUGP-PWT(ρ) with k labels it is NP-hard to distinguish whether $Val(G) > 1 - \zeta$ or $Val(G) < \delta$.*

Conjecture 6. (Unique Game Conjecture on Min GUGP-PWT(ρ)) *For every $\zeta, \delta > 0$, there is a $k = k(\zeta, \delta)$ such that given an instance of Min GUGP-PWT(ρ) with k labels it is NP-hard to distinguish whether $Val(G) < \zeta$ or $Val(G) > 1 - \delta$.*

The conjectures states it is NP-hard to distinguish the following two cases: there is a labeling under which the absolute value of the total of the negative weight of the unsatisfied edges is almost no less than the total of the positive weight of the unsatisfied edges; under any labeling the absolute value of the total of the negative weight of the satisfied edges is almost no less than the total of the positive weight of the satisfied edges.

3 GUGP-NWA

In this section, we prove it is NP-hard to approximate Min GUGP-NWA within any factor of $poly(n)$, and we prove Max GUGP-NWA can be approximated with factor 2.

Theorem 1. *It is NP-hard to approximate Min GUGP-NWA within any factor of $poly(n)$.*

Proof. Min GUGP-NWA can be restated as: In the situation of UGP, the goal is to find minimum fraction of the total weight of the *satisfied edges*. We construct an approximation ratio preservation reduction from TSP to the above problem.

Given an instance of TSP problem $G = (V, E)$, where each edge of E has a weight $w_e \in \mathbb{Q}^+$. Denote $n := |V|$. The instance of the restated form of Min GUGP-NWA is a graph $G' = G'(V, E')$, with each edge $e' \in E'$ having a weight $w'(e')$, and with the labeling set $[n]$. For each edge $e = (u, v) \in E$, there are three parallel edges $e^=$, e^+ and e^- between u and v in E'. $e^=$ has weight M and equipped with permutation $\pi^= = \{(1, 1), (2, 2), \cdots, (n, n)\}$. Let $M = n \cdot Max(w)$, where $Max(w)$ is the maximum weight of all edges in G. e^+ has weight $w_(e)$ and equipped with permutation $\pi^+ = \{(1, 2), (2, 3), \cdots, (n, 1)\}$. e^- has weight $w_(e)$ and equipped with permutation $\pi^- = \{(1, n), (2, 1), \cdots, (n, n-1)\}$.

Given a solution of TSP problem, a Hamiltonian cycle C, we can assign label 1 to n to vertices of C along C in G', and the total weight of satisfied edges in G' is exactly the total weight of edges on C in G.

In the other direction, given a labeling f of G', if there are two vertices assigned with the same label, the total weight of the satisfied edges is at least M. Otherwise all vertices are assigned with label from 1 to n respectively, let u_i be the vertices assigned label i for $1 \le i \le n$, and $e_i^+ \in E'$ be the edge between u_i and $u_{i+1 \ mod \ n}$ equipped with permutation π^+. The total weight of the satisfied edges is equal to $\sum_{1 \le i \le n} w'(e_i^+)$. Let C be the Hamiltonian cycle of G which consists of vertices from u_1 to u_n, then the total weight of C in G is exactly the total weight of satisfied edges under f in G'. □

Theorem 2. *Max GUGP-NWA can be approximated within factor 2.*

Proof. Max GUGP-NWA can be restated as the following 2P1R. We are given an graph $G = (V, E)$, a weight function $w_e \in \mathbb{Q}^+$, and the set of labels, $[k]$. Each edge $e = (u, v)$ in the graph is equipped with a relation $\bar{\pi}_e = [k] \times [k] - \pi_e$, where $\pi_e : [k] \to [k]$ is a permutation. The solution of the problem is a labeling $f : V \to [k]$ which assigns a label to each vertex of G. An edge $e = (u, v)$ is said to be satisfied under f if $(f(u), f(v)) \in \bar{\pi}_e$. The value of the instance is defined as the maximum fraction of the total weight of the satisfied edges.

We describe an approximation algorithm that finds a solution under which the fraction of the total weight of the satisfied edges is at least $1/2$, which is at least half of the value of the instance.

In the beginning of the algorithm, assign arbitrary labels to all vertices. Let $Ass(v, e)$ be the predicate whether $v \in V$ is associated with $e \in E$, and $Sat(e)$ be the predicate whether e is satisfied by current labeling. In each iteration of the algorithm, let $U^< = \{v \in V | \sum_{Ass(v,e) \wedge Sat(e)} w_e < \frac{1}{2} \sum_{Ass(v,e)} w_e\}$, and $U^\ge = \{v \in V | \sum_{Ass(v,e) \wedge Sat(e)} w_e \ge \frac{1}{2} \sum_{Ass(v,e)} w_e\}$. If $U^< = \emptyset$, the algorithm stops. Otherwise, choose a vertex u from $U^<$, suppose the label assigned to u is f_1, assign another label f_2 to the vertex. If an edge $e = (u, v)$ is unsatisfied under the old labeling, it must be the case $(f_1, f(v)) \notin \bar{\pi}_e$, and $\pi_e(f_1) = f(v)$. So $\pi_e(f_2) \ne f(v)$, and $(f_2, f(v)) \in \bar{\pi}_e$. Therefore, in the new labeling vertex u satisfies the condition of $U^\ge$, and we move it from $U^<$ to $U^\ge$. Since after each iteration, the number of vertices in $U^\ge$ is increased by 1, the algorithm stops in $|V|$ iterations. □

4 GUGP-PWT(ρ)

4.1 Parallel Repetition of Max 3-Cut

In Max 3-Cut Problem, the instance is a graph $G = (V, E)$, the value of the instance is the maximum fraction of properly colored edges of G under a 3-coloring. 3-coloring of a graph in a color set [3] is a function $\chi : V \to [3]$. We say an edge is properly colored under a 3-coloring if its endpoints receive distinct colors. A graph is 3-colorable if there is a coloring under which all edges are proper colored. Max 3-Cut Problem can be viewed as a 2P1R with $k = 3$, where each edge is equipped with a relation $\pi_e = \{(1, 2), (2, 3), (3, 1), (2, 1), (3, 2), (1, 3)\}$.

Given an instance of Max 3-Cut Problem, G, we define the l-fold parallel repetition of the instance , $G^l = G^l(V^l, E^l)$, as follows. $G^l = \{< u_1, \cdots, u_l > |u_i \in V, 1 \le i \le l\}$, and $E^l = \{(< u_1, \cdots, u_l >, < v_1, \cdots, v_l >)|(u_i, v_i) \in E, 1 \le i \le l\}$. The value of the instance is the maximum fraction of properly colored edges under a l-fold 3-coloring. Define a l-fold 3-coloring of G^l in the color set $[3]^l$ be the function $\chi^l : V \to [3]^l$, and let $m(\chi^l)$ be the number of properly colored edges under χ^l. We say an edge $e = (< u_1, \cdots, u_l >, < v_1, \cdots, v_l >)$ in E^l is properly colored under a l-fold 3-coloring if $\chi(u_i) \neq \chi(v_i)$ for any $1 \le i \le l$. The graph G^l is 3-colorable if there is a l-fold 3-coloring under which all edges are properly colored.

Petrank [11] shows that Max 3-Cut Problem possesses a hard gap at location 1, i.e. it is NP-hard to distinguish whether the instance is 3-colorable or whether has value at most $1 - \gamma$ for some constant γ. The constant is presumably very small and not determined in his paper. V. Guruswami et al. [6] make the constant clear to be $1/33 - \alpha$ for any $\alpha > 0$.

Lemma 1. ([11] Theorem 3.3) *It is NP-hard to distinguish whether the instance of Max 3-Cut Problem is whether 3-colorable or has value at most* $1-\gamma$ *for some constant* $\gamma > 0$.

Raz's Parallel Repetition Theorem is used to enlarge the gap of 2P1R with perfect completeness. We introduce Lemma 2 when applying Parallel Repetition Theorem to l-fold parallel repetition of Max 3-Cut Problem, and get Lemma 3 by a gap-reduction.

Lemma 2. ([12] Theorem 1.1) *If an instance of Max 3-Cut Problem has value* $1 - \gamma$*, the value of the l-fold parallel repetition of the instance is at most* $(1 - \gamma^{c_1})^{c_2 l}$*, where* c_1 *and* c_2 *are two positive constants.*

Lemma 3. *For any constant* $\delta > 0$*, there is a constant* $l = l(\delta)$ *such that it is NP-hard to distinguish whether the instance of l-fold parallel repetition of Max 3-Cut Problem is l-fold 3-colorable or has value at most* δ.

Proof. Given an instance of Max 3-Cut Problem, $G = (V, E)$, let l be the integer no less than $\frac{\ln \delta}{c_2 \ln(1-\gamma^{c_1})}$. Let G^l be the l-fold parallel repetition of G. The proof can be achieved by the following two steps and Lemma 2.

Completeness. Suppose G is 3-colorable. Define a l-fold 3-coloring of G^l as $\chi^l(< v_1, \cdots, v_l >) =< \chi^*(v_1), \cdots, \chi^*(v_l) >$, where χ^* is the optimal 3-coloring of G. Since all edges in G are properly colored under χ^*, all edges in G^l are properly colored under χ^l. Therefore, G^l is l-fold 3-colorable.

Soundness. Suppose G has value at most $1-\gamma$. By Lemma 2, the value of G^l is at most $(1-\gamma^{c_1})^{c_2 l}$, which is at most δ by the definition of l. □

4.2 Unique Game Conjecture on GUGP-PWT(ρ)

Let us show Conjecture 6 holds true at the boundary of the range of ρ.

Theorem 3. *Conjecture 6 holds true for $\rho = 1$.*

Proof. Given $\zeta, \delta > 0$, let $G^l = (V^l, E^l)$ be an instance of the parallel repetition of the Max 3-Cut Problem. The instance of Min GUGP-PWT(1) is a graph $G' = (V^l, E')$, with labeling set $[3^l]$.

To accomplish the reduction, we design a gadget as replacing each edge $e = (u, v)$ in E^l with 3^l parallel edges $e_{i_1}, \cdots, e_{i_l}$ for $1 \le i_j \le 3$, $1 \le j \le l$ between u and v in E'. Let $E^=$ be the set of edges such that at least one index is 1, and $E^{\neq}$ be the set of edges such that all indexes are 2 or 3. Note that $|E^=| = 3^l - 2^l$ and $E^{\neq} = 2^l$. Edges in $E^=$ has weight w_x, and edges in $E^{\neq}$ has weight w_y. $e_{i_1,\cdots,i_l}$ for $1 \le i_j \le 3$, $1 \le j \le l$ is equipped with permutation $\pi_{i_1,\cdots,i_l} = \{(<f_1, \cdots, f_l>, <f_1 + i_1 - 1 \ mod\ 3, \cdots, f_l + i_l - 1 \ mod\ 3>) | f_j \in [3], 1 \le j \le l\}$.

Note that there is always exactly one satisfied edge in E' between u and v under any labeling of G'. Suppose two vertices u and v are assigned labels $<f_{u,1}, \cdots, f_{u,l}>$ and $<f_{v,1}, \cdots, f_{v,l}>$ respectively. We require: (i)when $f_{u,j} = f_{v,j}$ for at least one $1 \le j \le l$, the total weight of the unsatisfied edges between u and v in E' is 1; (ii)when $f_{u,j} \neq f_{v,j}$ for any $1 \le j \le l$, the total weight of the unsatisfied edges between u and v in E' is 0.

Let us determine the value of w_x and w_y. By the linear equations,

$$\begin{cases} (3^l - 2^l - 1)w_x + 2^l w_y = 1 \\ (3^l - 2^l)w_x + (2^l - 1)w_y = 0 \end{cases},$$

we have

$$\begin{cases} w_x = -\dfrac{2^l - 1}{3^l - 1} \\ w_y = \dfrac{3^l - 2^l}{3^l - 1} \end{cases}.$$

Note that $w_x < 0$ and $w_y > 0$. $W^+_{G'} = \frac{2^l(3^l-2^l)}{3^l-1}|E^l|$, $W^-_{G'} = -\frac{(2^l-1)(3^l-2^l)}{3^l-1}|E^l|$, $\Sigma_{G'} = \frac{3^l-2^l}{3^l-1}|E^l|$, and $r_{G'} = |W^-_{G'}|/W^+_{G'} = 1 - \frac{1}{2^l} = 1 - O(\delta^c)$, where c is a positive constant. $r_{G'} \to 1$ when $\delta \to 0$.

The proof is completed by the following two steps and Lemma 3.

Completeness. Suppose G^l is l-fold 3-colorable. Let χ^l be the optimal l-fold 3-coloring, then $m(\chi^l) = |E^l|$. Let $f = \chi^l$. For any edge $e = (u, v)$ in E^l, $f_{u,j} \neq f_{v,j}$ for any $1 \le j \le l$, since χ^l is a l-fold 3-coloring. So the total weight of the unsatisfied edges between u and v is 0. Therefore, the total weight of the unsatisfied edges in E' is 0, i.e. $Val(G') = 0 < \zeta$.

Soundness. Suppose the value of G^l is at most δ. For any labeling f of G', $\chi^l = f$ is a l-fold 3-coloring of G^l, and $m(\chi^l) < \delta$. So at least $1 - \delta$ fraction of edges in E^l are not properly l-fold 3-colored. The two vertices of these edges share the same color in at least one element, in another word, the labels of the two vertices under f share the same value in at least one element. Since the total weight of the unsatisfied edges between such two vertices in E' is 1, the total weight of the unsatisfied edges in E' under f is at least $(1 - \delta)|E^l|$. Therefore, $Val(G') \geq (1 - \delta)|E^l|/\Sigma_{G'} > 1 - \delta$. □

We prove that Conjecture 6 holds true for $\rho = 1/2$ if Conjecture 4 holds true. Since Conjecture 3 implies Conjecture 4, Conjecture 6 holds true for $\rho = 1/2$ if Conjecture 3 holds true.

For any two positive integers m and n, let r be the reminder when dividing m by n, then $0 \leq r \leq n - 1$. Let $m\ mod^+\ n = r,\ if\ r > 0;\ n,\ if\ r = 0$.

Theorem 4. *Conjecture 6 holds true for* $\rho = 1/2$ *if Conjecture 4 holds true.*

Proof. Given $\zeta, \delta > 0$, let $G = (V, E)$ be an instance of 2-to-2 Game Problem, with labeling set $[2k]$. We construct an instance of Min GUGP-PWT(1/2) as a graph $G' = (V, E')$, with labeling set $[2k]$. For each edge $e = (u, v)$ in E with the 2-to-2 relation R, let the two permutations w.r.t. R are π_u, π_v, we design a gadget as replacing e with $2k$ parallel edges $e_1, \cdots, e_{2k}$ between u and v in E'.

The edge e_1 has weight w_x and is equipped with the permutation

$$\pi_1 = \{(\pi_u^{-1}(1), \pi_v^{-1}(1)), (\pi_u^{-1}(2), \pi_v^{-1}(2)), \cdots, (\pi_u^{-1}(2k-1), \pi_v^{-1}(2k-1)), (\pi_u^{-1}(2k), \pi_v^{-1}(2k))\}.$$

The edge e_2 has weight w_x and is equipped with the permutation

$$\pi_2 = \{(\pi_u^{-1}(1), \pi_v^{-1}(2)), (\pi_u^{-1}(2), \pi_v^{-1}(1)), \cdots, (\pi_u^{-1}(2k-1), \pi_v^{-1}(2k)), (\pi_u^{-1}(2k), \pi_v^{-1}(2k-1))\}.$$

The edge e_{2j-1} for $2 \leq j \leq k$ has weight w_y and is equipped with the permutation

$$\pi_{2j-1} = \bigcup_{i=1}^{k} \{(\pi_u^{-1}(2i-1), \pi_v^{-1}(2i - 1 + 2j - 2\ mod^+\ 2k)), (\pi_u^{-1}(2i), \pi_v^{-1}(2i + 2j - 2\ mod^+\ 2k\ ))\}.$$

The edge e_{2j} for $2 \leq j \leq k$ has weight w_y and is equipped with the permutation

$$\pi_{2j} = \bigcup_{i=1}^{k} \{(\pi_u^{-1}(2i-1), \pi_v^{-1}(2i - 1 + 2j - 1\ mod^+\ 2k\ )), (\pi_u^{-1}(2i), \pi_v^{-1}(2i + 2j - 3\ mod^+\ 2k\ ))\}.$$

Note that there is always exactly one satisfied edge in E' between u and v under any labeling of G'. Suppose two vertices u and v are assigned labels f_u and f_v respectively. We require: (i)when the edge $e = (u, v)$ is satisfied under f in G, i.e. one of the two edges e_1 and e_2 is is satisfied, the total weight of the unsatisfied edges between u and v in E' is 1; (ii)when when the edge $e = (u, v)$ is unsatisfied under f in G, i.e. one of the edges e_{2j-1} and e_{2j} for $2 \leq j \leq k$ is satisfied, the total weight of the unsatisfied edges between u and v in E' is 0.

Let us determine the value of w_x and w_y. By the linear equations

$$\begin{cases} w_x + (2k-2)w_y = 0 \\ 2w_x + (2k-3)w_y = 1 \end{cases},$$

we have

$$\begin{cases} w_x = \dfrac{2k-2}{2k-1} \\ w_y = -\dfrac{1}{2k-1} \end{cases}.$$

Note that $w_x > 0$, $w_y < 0$. $W^+_{G'} = \frac{4(k-1)}{2k-1}$, $W^-_{G'} = \frac{-2(k-1)}{2k-1}$, $\Sigma_{G'} = \frac{2k-2}{2k-1}|E|$ and $r_{G'} = 1/2$. The proof is completed by the following two steps.

Completeness. Suppose $Val(G) = 0$. Let f be the optimal labeling of G, then f is also a labeling of G'. Since any edge $e = (u, v)$ in E is satisfied under f, the total weight of the unsatisfied edges between u and v in E' is 0. Therefore, the total weight of the unsatisfied edges in E' is 0, i.e. $Val(G') = 0 < \zeta$.

Soundness. Suppose $Val(G) > 1-\delta$. Then for any labeling f of G', f is a labeling of G, at least $1-\delta$ fraction of the edges in E are unsatisfied. By the definition of E', the total weight of the unsatisfied edges in E' between the two endpoints of such edges is 1. So the total weight of the unsatisfied edges in E' under f is at least $(1-\delta)|E|$. Therefore, $Val(G') \geq (1-\delta)|E|/\Sigma_{G'} > 1-\delta$. □

In the end of this section, we establish a connection from Conjecture 6 to the Unique Game Conjecture by the following theorem.

Theorem 5. *If Conjecture 6 holds true for any $\rho > 0$, Conjecture 2 holds true.*

Proof. Suppose Conjecture 2 holds false, then for some $\zeta, \delta > 0$, for any label size k, we can decide in polynomial time whether an instance of Min UGP with k labels has a value more than $1-\delta$ or less than ζ. We claim Conjecture 6 for $\rho = \min(\zeta, \delta)/2$ holds false.

Given an instance $G = (V, E)$ of Min GUGP-PWT(ρ), we construct an instance $G' = (V, E')$ of Min UGP as follows. Let E' be the set of the edges in E with positive weights. Let f^* be the optimal labeling of G, and f' be the optimal labeling of G'. Then $Val(G') = W_{G'}(f')/W^+_G$ and $Val(G) = W_G(f^*)/\Sigma_G$.

Since $W_{G'}(f') \geq W_G(f') \geq W_G(f^*)$ and $\Sigma_G/W^+_G \geq 1-\rho$, $Val(G') \geq (1-\rho)Val(G)$.

By the definition of E', $W_G(f^*) \geq W_{G'}(f^*) - \rho W^+_G$. We have $Val(G)\Sigma_G = W_G(f^*) \geq W_{G'}(f^*) - \rho W^+_G \geq W_{G'}(f') - \rho W^+_G = (Val(G') - \rho)W^+_G$. Therefore, $Val(G') \leq Val(G) + \rho$.

If $Val(G) < \zeta/2$, then $Val(G') < \zeta$. If $Val(G) > 1-\delta/2$, then $Val(G') > 1-\delta$. Thus we can decide in polynomial time whether the instance of Min GUGP-PWT(ρ) has a value more than $1-\delta/2$ or less than $\zeta/2$. □

5 Discussions

The topic in this paper is similar to Khot's smoothness property in that both discuss on an analogue of UGP with proven $(1-\zeta, \delta)$-gap hardness. It leaves as an open problem whether Unique Game Conjecture holds true on GUGP-PWT(ρ) for $0 < \rho < 1$. We make a reasonable and rather bold conjecture: if Conjecture 6 holds true for some $0 < \rho < 1$, it holds true for any $0 < \rho < 1$, which would lead to the corollary that the d-to-1 Conjecture implies the Unique Game Conjecture, by Theorem 4 and Theorem 5. To confirm our conjecture, it would be very interesting to seek techniques to derive the $(1 - \zeta, \delta)$-gap hardness result on smaller ρ by the hardness result on larger ρ.

References

1. Khot, S.: On the power of unique 2-prover 1-round games. In: 34th Annual ACM Symposium on Theory of Computing, pp. 767–775 (2002)
2. Rao, A.: Parallel repitition in projection games and a concentration bound. In: 38th Annual ACM Symposium on Theory of Computing, pp. 1–10 (2008)
3. Khot, S., Kindler, G., Mossel, E., O'Donnell, R.: Optimal inapproximability results for Max-Cut and other 2-variable CSPs? In: 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 146–154 (2004)
4. O'Donnell, R., Wu, Y.: Conditional Hardness for satisfiabl CSPs. In: 39th Annual ACM Symposium on Theory of Computing, pp. 493–502 (2009)
5. Dinur, I., Mossel, E., Regev, O.: Conditional hardness for approximate coloring. In: 38th Annual ACM Symposium on Theory of Computing, pp. 344–353 (2006)
6. Guruswami, V., Sinop, A.K.: Improved Inapproximability Results for Maximum k-Colorable Subgraph. In: 12th International Workshop and 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, pp. 163–176 (2009)
7. Khot, S.: Hardness results for coloring 3-colorable 3-uniform hypergraphs. In: 43th Annual IEEE Symposium on Foundations of Computer Science, pp. 23–32 (2002)
8. Holmerin, J., Khot, S.: A new PCP outer verifier with applications to homogeneous linear equations and max-bisction. In: 38th Annual ACM Symposium on Theory of Computing, pp. 11–20 (2006)
9. Khot, S., Saket, R.: On hardness of learning intersections of two halfspaces. In: 40th Annual ACM Symposium on Theory of Computing, pp. 345–354 (2008)
10. Guruswami, V., Raghavendra, P., Saket, R., Wu, Y.: Bypassing UGC from some optimal geometric inapproximability results. Electronic Colloquium on Computational Complexity, TR10-177 (2010)
11. Petrank, E.: The hardness of approximation: gap location. Computational Complexity 4, 133–157 (1994)
12. Raz, R.: A parallel repitition theorem. SIAM Journal on Computing 27, 763–803 (1998)

A Note on Treewidth in Random Graphs[⋆]

Chaoyi Wang[1], Tian Liu[1], Peng Cui[2], and Ke Xu[3]

[1] Key Laboratory of High Confidence Software Technologies, Ministry of Education, Institute of Software, School of Electronic Engineering and Computer Science, Peking University, Beijing 100871, China
{wchaoyi,lt}@pku.edu.cn

[2] Key Laboratory of Data Engineering and Knowledge Engineering, MOE, School of Information Resource Management, Renmin University of China, Beijing 100872, China
cuipeng@ruc.edu.cn

[3] National Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
kexu@nlsde.buaa.edu.cn

Abstract. We show that in Erdős-Rényi random graph $G(n,p)$ with high probability, when $p = c/n$ and c is a constant, the treewidth is upper bounded by tn for some constant $t < 1$ which may depend on c, but when $p \gg 1/n$, the treewidth is lower bounded by $n - o(n)$. The upper bound refutes a conjecture that treewidth in $G(n, p = c/n)$ is as large as $n - o(n)$, and the lower bound provides further theoretical evidence on hardness of some random constraint satisfaction problems called Model RB and Model RD.

1 Introduction

Treewidth is an important notion with many applications in both combinatoric researches and algorithmic researches, see e.g. [2,16]. It measures how different a graph is from a tree. The smaller the treewidth is, the more similar the graph is to a tree. Many $\mathcal{NP}$-hard problems become tractable when restricted to trees or to graphs with a bounded treewidth. For example, constraint satisfaction problems (CSPs) are solvable in $O(||I||^{tw})$ time, where $||I||$ is the input size and tw is the treewidth of the constraint graph [5]. To investigate the tractability of random CSPs and Bayesian networks, Gao initiated the study of treewidth in various kinds of random (hyper)graphs from the considerations of random CSPs [8,9,10]. The most classical random graphs $G(n,p)$ are defined on n vertices by selecting each pair of vertices independently at random with probability p as an edge [12]. They are also called the Erdős-Rényi random graphs and have been extensively investigated, see e.g. [1,14,19]. In this paper, we show upper and lower bounds on treewidth in $G(n,p)$ for $p = O(1/n)$ and $p \gg 1/n$ respectively, also from a consideration of providing theoretical evidence on hardness of random CSPs.

[⋆] To whom the correspondence should be addressed: Tian Liu (lt@pku.edu.cn) and Ke Xu (kexu@nlsde.buaa.edu.cn).

Much earlier than Gao's works and even earlier than Robertson and Seymour introduced the notion of treewidth in 1984 [20], in the very beginning days of random graphs in 1959-1960, Erdős and Rényi had showed that each connected component in $G(n, p)$ has at most one cycle when $p = c/n$ and $c < 1$ [6,7], thus the treewidth is no more than two with high probability in that case. In a monograph on treewidth published in 1994, Kloks showed that treewidth is $\Theta(n)$ in $G(n, p)$ with high probability, when $p = c/n$ and $c > 2.36$ [16]. To close the gap between the case of bounded treewidth when $c < 1$ and the case of linear treewidth when $c > 2.36$ in $G(n, p)$, in a serie of works during the years 2003-2009, by the same method based on the notion of balanced partition as Kloks used, Gao reduced the linear treewidth threshold from $c > 2.36$ [16,8] down to $c > 2.162$ [9] and $c > 2.146$ [10], and conjectured that it is less than 2. Finally at 2010, the gap was closed completely when Lee et al. showed that the linear treewidth threshold is indeed $c > 1$, by a different method based on rankwidth [17].

Thanks to the above works, now it is clear that in $G(n, p = c/n)$ with high probability, treewidth is no more than two when $c < 1$ and is about tn when $c > 1$. However, an exact relationship between the constants t and c is unknown. In particular, it is not known when the treewidth is as large as $n - o(n)$. Since t is increasing with c and arbitrarily close to 1 for large enough c, one might conjecture that for some large constant c, the treewidth is as large as $n - o(n)$. In this short note, we show that this conjecture is false. Specifically, we show that in $G(n, p)$ with high probability, treewidth is less than tn for some constant $t < 1$ which may depend on c, when $p = c/n$ and c is a positive constant. The upper bound is obtained by an explicit construction of a specific tree decomposition. We also show that with high probability treewidth in $G(n, p)$ is larger than $n - o(n)$ when $p \gg 1/n$, along with some upper and lower bounds of t in terms of c. The lower bound is obtained by the same method as Kloks and Gao used in their works.

We can apply the above lower bound to provide further theoretical evidence on hardness of some random CSPs, called Model RB and Model RD. Since the discovering of satisfiability phase transition in random CSPs and the complexity peak of random instances around the satisfiability threshold [4], random instances are used as benchmarks in algorithm competitions and researches, see e.g. [13]. A whole family of benchmarks have been constructed [22,24] based on two random CSP models called Model RB and Model RD which have exact satisfiability thresholds [21]. These benchmarks cover many $\mathcal{NP}$-hard optimization problems, such as Maximum Clique, Maximum Independent Set, Minimum Vertex Cover, Vertex Coloring, Set Covering, Set Packing, 0-1 Integer Programming, MAX-SAT, etc.. In many international algorithm competitions, such as the annual CSP solver competitions and the annual Pseudo-Boolean (0-1 Integer Programming) solver competitions since the year 2005, the annual MAX-SAT solver competitions since the year 2008, and the annual SAT solver competitions at the years 2004 and 2009, etc., and in many research papers on algorithms, e.g. [3] and references in [22], these benchmarks have been successfully applied.

However, a rigorous link between phase transition and the hardness of random instance around the satisfiability threshold is still unknown. In the past, the main theoretical evidence on hardness of Model RB and Model RD was based on high resolution complexity [23]. Recently, some results on large structural width provided further theoretical evidences on hardness of Model RB and Model RD, such as large loop cueset [15] and large hinge width [18]. In most structural decomposition methods such as the tree decomposition based CSP algorithm, the first step is finding a decomposition of the given CSP instance, the second step is performing a join operation on constraint relations contained in each node of the decomposition, to formulate a new solution-equivalent tree-like CSP instance, then the third step is solving this tree-like instance [5]. Each decomposition of a CSP instance of large (i.e. unbounded) width contains a node with a large number of variables. Performing a join on all the variables in such a node is typically of high computational cost. Therefore, a large structural width around the satisfiability threshold can provide some theoretical evidences for these random instances to be hard for that kind of structural decomposition based algorithm. In this paper, we show that Model RB and Model RD have the largest possible treewidth $n - o(n)$ around their satisfiability threshold, thus provide further theoretical evidence on hardness of Model RB and Model RD with respect to the tree decomposition based algorithm.

This paper is organized as follows. In Section 2, we give one of the many equivalent definitions of treewidth, a definition of Model RB and Model RD and their random constraint graphs, and necessary notations and known results in probability theory. In Section 3, we show an upper bound tn with constant $t < 1$ on treewidth in $G(n, p)$ when $p = c/n$ and $c > 1$ is a constant. In Section 4, we show a lower bound $n - o(n)$ on treewidth in $G(n, p)$ when $p \gg 1/n$. In Section 5, we apply the lower bound result to show theoretical evidence on hardness of Model RB and Model RD. Finally, we discuss some open problems.

2 Preliminaries

Definition 1. *[16,8] A* tree decomposition *of a graph $G = (V, E)$ is a pair (T, χ), where $T = (N, F)$ is a tree, and χ is a labeling function associating to each vertex $p \in N$ a set of vertices $\chi(p) \subseteq V$, such that the following conditions are satisfied:*

1. *$\bigcup_{p \in N} \chi(p) = V$;*
2. *for all edges $(u, v) \in E$, there exists a node $p \in N$ such that both u and v are in $\chi(p)$;*
3. *for each vertex $v \in V$, the set of nodes $\{p \in N | v \in \chi(p)\}$ induces a subtree of T.*

The width *of the tree decomposition is* $\max_{p \in N} |\chi(p)| - 1$. *The* treewidth *of G is the minimum width over all its tree decompositions. We will denote the* treewidth *of a graph G as $tw(G)$.*

Definition 2. *[16,8]. Let G be a graph with $|V(G)| = n$ vertices. A partition (S, A, B) of $V(G)$ is called a* balanced w-partition *if the following three conditions are satisfied:*

1. $|S| = w + 1$;
2. $(n - w - 1)/3 \leq |A|, |B| \leq 2(n - w - 1)/3$;
3. *there are no edges connecting vertices of A and vertices of B.*

It is well-known that if a graph has a tree width no greater than w, then it must has a balanced w-partition.

Definition 3. *[21] Model RB and Model RD are defined as follows.*

- *Given n variables each with domain $\{1, 2, ..., d\}$, where $d = n^\alpha$ and $\alpha > 0$ is a constant;*
- *Select with repetition $m = rn \ln n$ random constraints, for each constraint select without repetition k of n variables, where $k \geq 2$ is an integer constant;*
- *Select uniformly at random without repetition $(1 - p)d^k$ compatible assignments for each constraint (for Model RB), or select each assignment for the k variables as compatible with probability $1-p$ independently (for Model RD), where $0 < p < 1$ is a constant.*

Let $G^{RB}(n, r)$ denote the probability space of the random constraint graphs of Model RB and Model RD for $k = 2$, where on n vertices we select with repetition $rn \ln n$ edges from all pairs of vertices independently at random.

It was known that in Model RB and Model RD there exist satisfiability thresholds $r_{cr} = -\frac{\alpha}{\ln(1-p)}$ [21].

Definition 4. *For events $\mathcal{Q}_n$ which may depend on n, we say that $\mathcal{Q}_n$ holds* with high probability, *if* $\lim_{n\to\infty} \Pr(\mathcal{Q}_n) = 1$, *often written as* $\Pr(\overline{\mathcal{Q}_n}) = o(1)$. *Here, $f(n) = o(g(n))$ and $f(n) \ll g(n)$ both mean that* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

Let $\mathbb{E}(X)$ be the expectation of a random variable X. We will use the following Chernoff Bound [14,19].

Lemma 1. *(*Chernoff Bound*) Let $X = \sum_{i=1}^{n} X_i$, where X_i are independent* 0-1 *variables. Let $\mu = \mathbb{E}(X)$. Then for any $\delta > 0$,* $\Pr(X \geq (1 + \delta)\mu_H) \leq e^{-\frac{\delta^2}{3}\mu}$.

3 The Upper Bound

In this section, we show that when $p = c/n$ and $c > 1$ is a constant, the treewidth in $G(n, p)$ is less than tn with high probability for some constant $t < 1$ which depends on c.

Theorem 1. *Let $G \in G(n, p = c/n)$, where $c > 1$ is a constant. Then for any $\delta > 0$ and $a = (1 + \delta)c$, with high probability $tw(G) \leq \frac{a+1}{a+2}n$.*

Proof. We will show that, with high probability we can construct a tree decomposition with the deserved property.

Consider a partition (A, B) of $V(G)$, with $|A| = x = tn$ and $|B| = n - tn$, where the constant $t \leq \frac{1}{2}$ will be specified later.

Let random variable X be the number of edges connecting vertices of A and vertices of B. The expectation of X is

$$\mathbb{E}(X) = x(n-x)p = t(1-t)cn.$$

By the Chernoff bound, for any $\delta > 0$,

$$\Pr\left(X \geq (1+\delta)\mathbb{E}(X)\right) \leq e^{-\frac{\delta^2}{3}\mathbb{E}(X)} = e^{-\frac{\delta^2 t(1-t)c}{3}n} = o(1).$$

Thus with high probability,

$$X < (1+\delta)\mathbb{E}(X) = (1+\delta)t(1-t)cn.$$

Let S be the set of vertices in B which are connected with vertices in A. Then $(S, A, B \setminus S)$ is a partition of $V(G)$ and with high probability

$$|S| \leq X < (1+\delta)t(1-t)cn.$$

Without loss of generality, we may assume that

$$|S| = (1+\delta)t(1-t)cn,$$

since if $|S| < (1+\delta)t(1-t)cn$, then we can always move some vertices from A or B to S, such that the resulting $(S, A, B \setminus S)$ is still a partition of $V(G)$ and no edge connects the vertices in A and the vertices in $B \setminus S$.

Let $|A| = |B \setminus S|$, that is

$$tn = (n - tn) - (1+\delta)t(1-t)cn.$$

Let $a = (1+\delta)c$, then

$$t = \frac{a + 2 - \sqrt{a^2+4}}{2a} = \frac{2}{a + 2 + \sqrt{a^2+4}} > \frac{1}{a+2}.$$

Now $A \cup S$ and B is a tree decomposition of G with two nodes. Thus

$$tw(G) \leq max\{|A \cup S|, |B|\} - 1 = (1-t)n - 1 < \left(1 - \frac{1}{a+2}\right)n = \frac{a+1}{a+2}n.$$

□

4 The Lower Bound

In this section, we show that in $G(n, p)$ with high probability, the treewidth is larger than $n - o(n)$ when $p \gg 1/n$. Along the way, we get some relation between t and c.

Theorem 2. *Let $G \in G(n, p = c/n)$, where $1 < c \ll n$, c is either constant or depends on n. For any $0 < t < 1$, t may change with n, for any $\delta > 0$, if*

$$c > \frac{9(\ln 2 + 1 - \ln t + \delta)}{2t},$$

then $tw(G) \geq (1-t)n$ with high probability.

Proof. By a similar method as in [16,8].

Let $\mathcal{P}$ be the set of all partitions of $V(G)$ that satisfied the first two conditions in Definition 2. The value of w will be specified later. For any single partition $P = (S, A, B) \in \mathcal{P}$, define a random variable I_P as follows.

$$I_p = \begin{cases} 1, P \text{ is a balanced } w\text{-partition,} \\ 0, \text{otherwise.} \end{cases}$$

Then $I_P = 1$ if and only if there are no edges between A and B.

Let

$$\begin{aligned} |S| &= w + 1 = (1-t)n, \\ |A| &= x, \\ |B| &= tn - x. \end{aligned}$$

Since there are $|A| \cdot |B| = x(tn - x)$ possible edges between A and B, the expectation of I_P is

$$\mathbb{E}(I_P) = \Pr(I_P = 1) = (1-p)^{x(tn-x)}.$$

Since $1 - p < e^{-p} = e^{-\frac{c}{n}}$ and that for $\frac{tn}{3} \leq x \leq \frac{2tn}{3}$, $x(tn-x)$ is minimized at $x = \frac{tn}{3}$ or $x = \frac{2tn}{3}$, we have

$$\mathbb{E}(I_P) \leq e^{-px(tn-x)} \leq e^{-\frac{2ct^2 n}{9}}.$$

Let $I = \sum_{P \in \mathcal{P}} I_P$. Then $I = 0$ if and only if there is no balanced w-partition in G, and only if G has tree width greater than w. By the linearity of expectation and the union bound, the expectation of I is

$$\begin{aligned} \mathbb{E}(I) = \textstyle\sum_{P \in \mathcal{P}} \mathbb{E}(I_P) &\leq \tbinom{n}{tn} \textstyle\sum_{\frac{tn}{3} \leq x \leq \frac{2tn}{3}} \tbinom{tn}{x} e^{-\frac{2ct^2 n}{9}} \\ &\leq \left(\tfrac{en}{tn}\right)^{tn} 2^{tn} e^{-\frac{2ct^2 n}{9}} = e^{tn\left(-\frac{2ct}{9} + \ln 2 + 1 - \ln t\right)}. \end{aligned}$$

For any $\delta > 0$, if $c > \frac{9(\ln 2 + 1 - \ln t + \delta)}{2t}$, then

$$\Pr(I > 0) \leq \mathbb{E}(I) \leq e^{-\delta tn} = o(1).$$

That is, with high probability $I = 0$, or there is no balanced $(1-t)n$-partition in G. Thus with high probability $tw(G) \geq (1-t)n$. □

Corollary 1. *Let $G \in G(n, p = c(n)/n)$. If $c(n)$ is unbounded, then $tw(G) = n - o(n)$ with high probability.*

Proof. Let $f(n)$ be unbounded, $f(n) = o(n)$ and $5f(n) \leq c(n)$. If $c(n)$ is unbounded, then such an $f(n)$ do exist. Let $t(n) = \frac{\ln f(n)}{f(n)}$. Then for any $\delta > 0$, for large n,

$$\frac{9(\ln 2 + 1 + \ln f(n) - \ln\ln f(n) + \delta) f(n)}{2 \ln f(n)} \leq 5f(n) \leq c.$$

By Theorem 2, with high probability

$$tw(G) \geq \left(1 - \frac{\ln f(n)}{f(n)}\right) n = n - o(n).$$

The proof is finished. □

5 Application

In this section, we apply the lower bound in last section to provide further theoretical evidence on hardness of some random CSPs called Model RB and Model RD. Recall that Model RB and Model RD are defined as follows [21].

– Given n variables each with domain $\{1, 2, ..., d\}$, where $d = n^{\alpha}$ and $\alpha > 0$ is a constant;
– Select with repetition $m = rn \ln n$ random constraints, for each constraint select without repetition k of n variables, where $k \geq 2$ is an integer constant;
– Select uniformly at random without repetition $(1-p)d^k$ compatible assignments for each constraint (for Model RB), or select each assignment for the k variables as compatible with probability $1-p$ independently (for Model RD), where $0 < p < 1$ is a constant.

For simplicity, we only consider Model RB and Model RD for $k = 2$. Clearly, they have the same kinds of random constraint graphs. Let $G^{RB}(n, r)$ denote the probability space of these random graphs, where on n vertices we select with repetition $rn \ln n$ edges from all pairs of vertices independently at random.

As in many other random graph models, $G^{RB}(n, r)$ and $G(n, p)$ are in fact asymptotically equivalent [15,18].

Lemma 2. *([15,18]) Let $\mathcal{Q}_n$ be an arbitrary graph property, $r > 0$ and $p = (rn \ln n - r^2 \ln^2 n)/\binom{n}{2}$. Then $\Pr_{G^{RB}(n,r)}(\mathcal{Q}_n) \leq e\sqrt{rn \ln n}\, \Pr_{G(n,p)}(\mathcal{Q}_n)$. Moreover, if $\mathcal{Q}_n$ is monotone, then $\Pr_{G^{RB}(n,r)}(\mathcal{Q}_n) \leq 2 \Pr_{G(n,p)}(\mathcal{Q}_n)$.*

Note that in above lemma, we can also set $p = (rn \ln n)/\binom{n}{2}$, which will make no difference asymptotically.

Corollary 2. *The treewidth in $G^{RB}(n, r)$ is $n - o(n)$ for any constant $r > 0$ with high probability.*

Proof. Treewidth is a monotone increasing graph property. Thus by lemma 2 and Corollary 1, the corollary is proved. □

By Corollary 2, the random instances of Model RB and Model RD around the satisfiability thresholds $r_{cr} = -\frac{\alpha}{\ln(1-p)}$ [21] have asymptotically the largest possible treewidth $n-o(n)$. In the standard tree decomposition decomposition based CSP solvers, after finding a tree decomposition of the given CSP instance, the join operation is performed on constraint relations contained in each node of the decomposition, to formulate a new solution-equivalent tree-like CSP instance [5]. Performing a join on $n-o(n)$ variables in the largest node of the decomposition is typically of high computational cost. Therefore, a large treewidth around the satisfiability threshold can provide some theoretical evidence for these random instances to be hard for that kind of tree decomposition based algorithm. Thus the above results provide further theoretical evidence on the hardness of Model RB and Model RD, besides the known theoretical hardness evidences based on their exponential resolution complexity [23], large loop cueset [15] and large hinge width [18].

6 Open Problems

In $G(n,p)$ when $p = c/n$ and $c > 1$ is a constant, we know that the treewidth is between $t_1 n$ and $t_2 n$ for some constants $0 < t_1 < t_2 < 1$ which may depend on c. The exact relationships between t_1, t_2 and c are unknown. Especially, tighter lower bound on $t_1(c)$ and tighter upper bound on $t_2(c)$ are unknown.

Acknowledgments. We thank Professor Kaile Su for his encouragement and support. We also thank the unknown reviewers whose comments are very helpful to improve our presentations. This research was partially supported by the National 973 Program of China (Grant No. 2010CB328103), the National Natural Science Foundation of China (Grant Nos. 60725207 and 60973033), the Project of Knowledge Innovation Program (PKIP) of Chinese Academy of Sciences (Grant No. KJCX2.YW.W10), and was partially done while Tian Liu and Ke Xu were visiting KITPC at the Chinese Academy of Science.

References

1. Bollobás, B.: Random Graphs, 2nd edn. Cambridge University Press, Cambridge (2001)
2. Bodlaender, H.L.: Treewidth: Characterizations, applications, and computations. Technical Report UU-CS-2006-041, Institute of Information and Computing Sciences, Utrecht University (2006)
3. Cai, S., Su, K., Chen, Q.: Ewls: A new local search for minimum vertex cover. In: Proc of AAAI, pp. 45–50 (2010)
4. Cheeseman, P., Kanefsky, R., Taylor, W.: Where the really hard problems are. In: Proc. IJCAI, pp. 163–169 (1991)

5. Dechter, R.: Tractable structures for constraint satisfaction problems. Handbook of constraint programming, 209–244 (2006)
6. Erdős, P., Rényi, A.: On random graphs I. Publicationes Mathematicae 6, 290–297 (1959)
7. Erdős, P., Rényi, A.: On the evolution of random graphs. Publ. Math. Inst. Hungar. Acad. Sci. 5, 17–61 (1960)
8. Gao, Y.: Phase transition of tractability in constraint satisfaction and Bayesian network inference. In: Proc. UAI, pp. 265–271 (2003)
9. Gao, Y.: On the threshold of having a linear treewidth in random graphs. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 226–234. Springer, Heidelberg (2006)
10. Gao, Y.: Treewidth of Erdös-Rényi random graphs, random intersection graphs, and scale-free random graphs. CoRR abs/0907.5481 (2009)
11. Gao, Y., Culberson, J.: Consistency and random constraint satisfaction problems. J. Artif. Intell. Res. 28, 517–557 (2007)
12. Gilbert, E.N.: Random graphs. Annals of Mathematical Statistics 30, 1141–1144 (1959)
13. Gomes, C., Walsh, T.: Randomness and structures. Handbook of constraint programming, pp. 639–664 (2006)
14. Janson, S., Luczak, T., Rucinski, A.: Random Graphs. John Wiley and Sons, Chichester (2000)
15. Jiang, W., Liu, T., Ren, T., Xu, K.: Two hardness results on feedback vertex sets. In: Zhu, B. (ed.) FAW-AAIM 2011. LNCS, vol. 6681, pp. 233–243. Springer, Heidelberg (2011)
16. Kloks, T.: Treewidth. LNCS, vol. 842. Springer, Heidelberg (1994)
17. Lee, C., Lee, J., Oum, S.: Rank-width of random graphs. CoRR abs/1001.0461 (2010)
18. Liu, T., Lin, X., Wang, C., Su, K., Xu, K.: Large Hinge Width on Sparse Random Hypergraphs. In: Proc. of IJCAI (2011)
19. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, Cambridge (2005)
20. Robertson, N., Seymour, P.: Graph minors. III. Planar tree-width. J. Combin. Theory Ser. B 36(1), 49–64 (1984)
21. Xu, K., Li, W.: Exact phase transitions in random constraint satisfaction problems. J. Artif. Intell. Res. 12, 93–103 (2000)
22. Xu, K.: BHOSLIB: Benchmarks with Hidden Optimum Solutions for Graph Problems, http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm
23. Xu, K., Li, W.: Many hard examples in exact phase transitions. Theor. Comput. Sci. 355, 291–302 (2006)
24. Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: Random Constraint Satisfaction: Easy Generation of Hard (Satisfiable) Instances. Artif. Intell. 171, 514–534 (2007)

On the Two-Stage Stochastic Graph Partitioning Problem

Neng Fan[1], Qipeng P. Zheng[2], and Panos M. Pardalos[1]

[1] Center for Applied Optimization, Department of Industrial and Systems Engineering,
University of Florida, Gainesville, FL, USA
{andynfan,pardalos}@ufl.edu

[2] Department of Industrial and Management Systems Engineering,
West Virginia University, Morgantown, WV, USA
qipeng.zheng@mail.wvu.edu

Abstract. In this paper we introduce the two-stage stochastic graph partitioning problem and present the stochastic mixed integer programming formulation for this problem with finite explicit scenarios. For solving this problem, we present an equivalent integer linear programming formulation where some binary variables are relaxed to continuous ones. Additionally, for some specific graphs, we present a more simplified linear programming formulation. All formulations are tested on randomly generated graphs with different densities and different numbers of scenarios.

Keywords: Graph Partitioning, Stochastic Optimization, Integer Programming.

1 Introduction

The graph partitioning problem consists of partitioning the vertex set of a graph into several disjoint subsets so that the sum of weights of the edges between the disjoint subsets is minimized and it is an NP-complete combinatorial optimization problem.

Let $G = (V,E)$ be an undirected graph with a set of vertices $V = \{v_1, v_2, \cdots, v_N\}$ and a set of edges $E = \{(v_i, v_j) : \text{edge between vertices } v_i \text{ and } v_j, 1 \le i, j \le N\}$, where N is the number of vertices. The weights of the edges are given by a matrix $W = (w_{ij})_{N\times N}$, where $w_{ij}(> 0)$ denotes the weight of edge (v_i, v_j) and $w_{ij} = 0$ if no edge (v_i, v_j) exists between vertices v_i and v_j. This matrix is symmetric for undirected graphs G. Thus, the edge set can be expressed by $E = \{(v_i, v_j) : w_{ij} > 0, 1 \le i < j \le N\}$. In the following we also use $(i, j) \in E$ to denote that $(v_i, v_j) \in E$.

Let K be the number of disjoint subsets that we want to partition V into, and let $n_1, \cdots, n_K$ be the cardinalities of subsets of V and these numbers are assumed to be given before partitioning. Usually, K is chosen from $\{2, \cdots, N-1\}$. As shown in [3,4], the size n_k for subset k can be loosely chosen in a range. For equal partitioning, n_k is chosen as N/K. In this paper, we assume K is given and set n_k around the value N/K.

Let x_{ik} be the indicator that vertex v_i belongs to the kth subset if $x_{ik} = 1$ or not if $x_{ik} = 0$, and y_{ij} be the indicator that the edge (v_i, v_j) with vertices v_i, v_j are in different subsets if $y_{ij} = 1$ and v_i, v_j in the same subset if $y_{ij} = 0$. Thus, the objective function of graph partitioning to minimize the sum of weights of the edges between the disjoint subsets can be expressed as $\min\sum_{(i,j)\in E} w_{ij}y_{ij}$. The relation between x_{ik} and y_{ij} can be expressed as $y_{ij} = 1-\sum_{k=1}^{K} x_{ik}x_{jk}$. Therefore, the graph partitioning under weight matrix W and given sizes $n_1,\cdots,n_K$ of subsets can be expressed as

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in E} w_{ij}y_{ij}, \qquad (1)\\
s.t. \quad & \sum_{k=1}^{K} x_{ik} = 1, \sum_{i=1}^{N} x_{ik} = n_k,\\
& y_{ij} = 1-\sum_{k=1}^{K} x_{ik}x_{jk},\\
& x_{ik}\in\{0,1\}, y_{ij}\in\{0,1\},\\
& \quad i\in V,(i,j)\in E, k=1,\cdots,K.
\end{aligned}$$

Since no uncertainty is considered in this model, we usually call this *nominal graph partitioning problem*. As stated in [3,5], many mathematical programming methods, including linear programming, quadratic programming, and semidefinite programming, are used for this problem. As discussed in [5], the weights of edges in graph $G=(V,E)$ are always uncertain. In mathematical programming, two methods are always used to deal with such uncertainty. The robust optimization models for graph partitioning, as studied in [5], is to find out a best partitioning of vertex set V among all uncertain weights of edges in the worst case. In this paper, we will introduce the two-stage stochastic graph partitioning problem with finite explicit scenarios to deal with the uncertainty.

The *set of cut edges* or the *cut set* includes all edges with ends in different subsets after partitioning. The *two-stage stochastic graph partitioning problem* (TSGP) consists of finding a best partitioning of vertex set in two stages: taking some edges into the set of cut edges in the first stage with certain weights for edges in matrix W; assuming that there are totally S scenarios with weight matrix $C^s = (c_{ij}^s)_{N\times N}$ in scenario s of probability p_s, and the second stage in scenario s is to choose some edges into the set of cut edges for satisfying the requirements of partitioning. The objective is to minimize the total expected weight of edges in the set of cut over all scenarios. Under these requirements and descriptions, we formulate the two-stage stochastic graph partitioning problem as a stochastic mixed integer program (SMIP) [2].

Similarly, many combinatorial optimization problems have been extended to two-stage stochastic forms recently. The two-stage maximum matching problem is studied in [7] by an approximation algorithm, the minimum spanning tree problem is extended to two-stage forms in [6], and etc.

In this paper, we assume the distribution of weights has finite explicit scenarios. The rest of this paper is organized as follows: Section 2 presents the model for the two-stage stochastic graph partitioning problem; In Section 3, we present equivalent integer linear formulations; In Section 4, we present numerical experiments on randomly generated graphs with different numbers of scenarios; Section 5 concludes the paper.

2 The Model of the Two-Stage Stochastic Graph Partitioning Problem

The two-stage stochastic graph partitioning problem can be formally stated as follows: Given a graph $G = (V, E)$ with the first-stage edge weight matrix W and second-stage edge weight matrix $C^s = (c^s_{ij})_{N\times N}$ for $s = 1, \cdots, S$, and the probability p_s for scenario s, where S is the number of scenarios. The edge set E now is defined as $E = \{(v_i, v_j) : w_{ij} > 0 \text{ or } c^s_{ij} > 0 \text{ for some } s, j > i\}$, which means if $w_{ij} > 0$ or one of $c^s_{ij} > 0$, the edge (v_i, v_j) exists. That is, for edge $(i, j) \in E$, the first-stage weight is w_{ij} and the second stage weight is c^s_{ij} in scenario s with probability p_s. In addition, we are also given the number K of subsets that we want to partition V into and the cardinalities $n_1, \cdots, n_K$ of all subsets.

Remark 1. The weight matrix C^s for scenario s has the same prefigurements as W: no loops in the graph, i.e., $c^s_{ii} = 0$ for $i = 1, \cdots, N$; symmetrically, i.e., $c^s_{ij} = c^s_{ji}$ for $i, j = 1, \cdots, N$; nonnegativity, i.e., $c^s_{ij} \geq 0$ for $i, j = 1, \cdots, N$.

Remark 2. The probability p_s for $s = 1, \cdots, S$ and weight matrices C^s's are defined on a probability space $(\Omega, \mathscr{C}, \mathscr{P})$, where Ω is the sample space and can be chosen as nonnegative real space $\mathscr{R}^{N\times N}_+$, $\mathscr{C}$ is the set of subsets in Ω, and $\mathscr{P}$ is the probability measure. In this problem, we assume finite explicit scenarios.

The *two-stage stochastic graph partitioning problem* (TSGP) is to find a set of cut edges E^C with the minimum sum of weights so that the subsets satisfy the requirements at each scenario. Assume that E^0 is the set of cut edges chosen in the first-stage, and E^s is the chosen set of cut edges in the second-stage with respect to scenario s for $s = 1, \cdots, S$, the sets have the relations $E^0 \cup E^s$ is the set that can completely separate the vertices into K subsets with the requirement of cardinalities, and $E^0 \cap E^s = \emptyset$. In addition, the cuts $E^0, E^1, \cdots, E^S$ should have the minimum expected sum of weights

$$\sum_{(v_i,v_j)\in E^0} w_{ij} + \sum_{s=1}^{S} p_s \sum_{(v_i,v_j)\in E^s} c^s_{ij}.$$

For example, in Fig. 1, the weights for edge $(v_i, v_j) \in E$ at fist stage (w_{ij}) and second stage (c^s_{ij}) for two scenarios $(p_1 = 0.6, p_2 = 0.4)$ are shown. The problem is to find 3 subsets with cardinalities as $n_1 = 3, n_2 = 4, n_3 = 5$. By the STGP model, two edges $(2,4), (7,12)$ are selected into the cut set at the first stage, while at the second stage, edges $(3,7), (3,11), (6,8)$ are selected in the first scenario $s = 1$ and edges $(4,7), (5,7), (6,7), (8,9), (8,11), (8,12), (10,11), (11,12)$ are selected in the second scenario $s = 2$. Three subsets obtained for scenario $s = 1$ are $\{1,2,3\}, \{4,5,6,7\}$, $\{8,9,10,11,12\}$, while three subsets for scenario $s = 2$ are $\{9,10,12\}, \{4,5,6,8\}$, $\{1,2,3,7,11\}$.

Assume that $y_{ij} = 1$ denotes that (i, j) is chosen to E^0 and otherwise $y_{ij} = 0$, and $z^s_{ij} = 1$ denotes edge (i, j) is chosen to E^s in scenario s and otherwise $z^s_{ij} = 0$. Let $x^s_{ik} = 1$ denote that the ith vertex belongs to the kth subset and otherwise $x^s_{ik} = 0$. By these decision variables, the two-stage stochastic graph partitioning problem can be formulated as the following two-stage program:

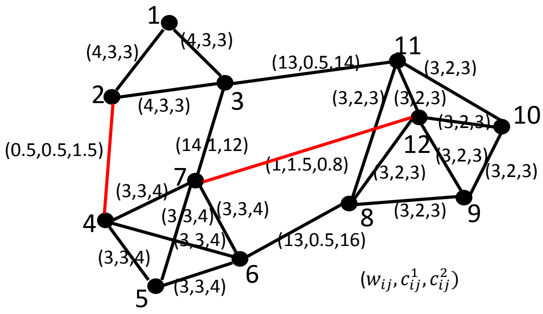


Fig. 1. A graph with uncertain edges

$$[\textbf{TSGP}] : \min \quad \sum_{(i,j)\in E} w_{ij} y_{ij} + \sum_{s=1}^{S} p_s f(y,s) \tag{2}$$

$$s.t. \quad y_{ij} \in \{0,1\}, \qquad i \in V, (i,j) \in E. \tag{3}$$

where for $s = 1, \cdots, S$,

$$f(y,s) = \min_{x,z} \quad \sum_{(i,j)\in E} c^s_{ij} z^s_{ij} \tag{4}$$

$$s.t. \quad \sum_{k=1}^{K} x^s_{ik} = 1, \sum_{i=1}^{N} x^s_{ik} = n_k \tag{5}$$

$$y_{ij} + z^s_{ij} = 1 - \sum_{k=1}^{K} x^s_{ik} x^s_{jk}, \tag{6}$$

$$x^s_{ik}, z^s_{ij} \in \{0,1\}, \qquad i \in V, (i,j) \in E, k = 1, \cdots, K. \tag{7}$$

Next, we first prove this formulation is the correct formulation for two-stage stochastic graph partitioning problem, and then discuss the relaxations of the variables y_{ij}'s and z^s_{ij}'s.

Theorem 1. *The formulation* (2)-(7) *is the correct model for the two-stage stochastic graph partitioning problem.*

Proof. From the objective function in (2), the decision variables y_{ij} and z^s_{ij} decide whether the edge (v_i, v_j) is included in the set of cut edges for scenario s with respect to the first stage weight w_{ij} and second stage weight c^s_{ij}, respectively. The constraints $x^s_{ik} \in \{0,1\}$ and the constraints (5) can guarantee that each vertex belongs to exact one subset and the kth subset has the cardinality n_k in the second stage of scenario s.

Thus, the set of cut edges in the first stage is $E^0 = \{(v_i, v_j) \in E : y_{ij} = 1\}$ and the set of cut edges in the second stage of scenario s is $E^s = \{(v_i, v_j) \in E : z^s_{ij} = 1\}$. We have to prove that $E^0 \cup E^s$ is the set of cut edges and $E^0 \cap E^s = \emptyset$ for any $s = 1, \cdots, S$.

If both vertex v_i and v_j belong to subset k in scenario s, i.e., $x^s_{ik} = x^s_{jk} = 1$ and $x^s_{ik'} = x^s_{jk'} = 0$ for $k' \neq k$, from the constraint (6), we have $y_{ij} + z^s_{ij} = 0$. Thus, $y_{ij} = z^s_{ij} = 0$ since both of them are binary. The edge (v_i, v_j) is not in the set of cut edges.

If the vertex v_i belongs to subset k_1 and v_j belongs to subset k_2 of scenario s, i.e., $x^s_{i,k_1} = 1, x^s_{ik} = 0$ for all $k \neq k_1$ and $x^s_{j,k_2} = 1, x^s_{j,k'} = 0$ for all $k' \neq k_2$ where $k_1 \neq k_2$, thus we have $y_{ij} + z^s_{ij} = 1$ from the constraint (6). Thus, since both $y_{ij}, z^s_{ij} \in \{0, 1\}$, either $y_{ij} = 1$, which means edge (v_i, v_j) is chosen in cut edges in the first stage, or $z^s_{ij} = 1$, which means (v_i, v_j) is chosen in the cut edges in the second stage of scenario s. Considering all edges, we have proved that $E^0 \cup E^s$ is the set of cut edges and $E^0 \cap E^s = \emptyset$ for any $s = 1, \cdots, S$.

The objective function is to minimize fist stage weight and the expected sum weight of all scenarios $s = 1, \cdots, S$ for edges within the cut. Therefore, we have checked the objective and all constraints, and the program (2)-(7) correctly formulates the two-stage stochastic graph partitioning problem. □

Corollary 1. *For edge $(v_i, v_j) \in E$, if $w_{ij} > \sum_{s=1}^S p_s c^s_{ij}$, we have $y_{ij} = 0$.*

Proof. By contradiction, if $y_{ij} = 1$, which means that edge (v_i, v_j) is chosen into the cut set in the first stage, and the objective with respect to this edge is w_{ij}. However, by choosing $c^s_{ij} = 1$ for all s, the corresponding objective is $\sum_{s=1}^S p_s c^s_{ij}$, which is less than w_{ij}, a contradiction to minimizing the objective. □

However, for the case $w_{ij} = \sum_{s=1}^S p_s c^s_{ij}$ for edge $(v_i, v_j) \in E$, if this edge is chosen into the cut set, either $y_{ij} = 0, c^s_{ij} = 1 (\forall s)$ or $y_{ij} = 1, c^s_{ij} = 0 (\forall s)$ are optimal solutions; if this edge is not chosen into the cut set, $y_{ij} = 0$. In order to reduce computation time, we make the following assumption without loss of any optimality:

Assumption A. For edge $(v_i, v_j) \in E$, if $w_{ij} = \sum_{s=1}^S p_s c^s_{ij}$, assume $y_{ij} = 0$.

3 Equivalent Integer Linear Programming Formulations

In the constraint (6), there is a nonlinear term $x^s_{ik} x^s_{jk}$, which always leads to high computational complexity. In this section, we present an approach to linearize this term. Additionally, we prove that some binary variables in the formulation for TSGP can be relaxed to continuous ones.

Lemma 1. *By Corollary 1 and under* Assumption A, *the decision variables y_{ij}, z^s_{ij} in the two-stage stochastic graph partitioning problem* (2)-(7) *can be relaxed to be continuous ones such that* $0 \leq y_{ij} \leq 1, z^s_{ij} \geq 0$.

Proof. In the part E^0, E^s of the proof for Theorem 1, in the case of vertices v_i, v_j in the same subset in scenario s, i.e., $y_{ij} + z^s_{ij} = 0$, we have $y_{ij} = z_{ij} = 0$ if $y_{ij}, z^s_{ij} \geq 0$.

In the case of vertices v_i, v_j in different subsets in scenario s, i.e., $y_{ij} + z^s_{ij} = 1$, we discuss in the following three cases:

(a) $w_{ij} > \sum_{s=1}^{S} p_s c_{ij}^s$. By Corollary 1, $y_{ij} = 0$ and then $z_{ij}^s = 1$.
(b) $w_{ij} = \sum_{s=1}^{S} p_s c_{ij}^s$. By *Assumption A*, $y_{ij} = 0$ and then $z_{ij}^s = 1$.
(c) $w_{ij} < \sum_{s=1}^{S} p_s c_{ij}^s$. We have $z_{ij}^s = 1 - y_{ij}$. The part in the objective function (2) with respect to edge (v_i, v_j) is

$$\begin{aligned}\min(w_{ij}y_{ij} + \sum_{s=1}^{S} p_s c_{ij}^s z_{ij}^s) &= \min(w_{ij}y_{ij} + \sum_{s=1}^{S} p_s c_{ij}^s (1 - y_{ij})) \\ &= \min(w_{ij} - \sum_{s=1}^{S} p_s c_{ij}^s) y_{ij} + \sum_{s=1}^{S} p_s c_{ij}^s.\end{aligned}$$

If $y_{ij} \in \{0,1\}$, the minimum for this problem is obtained when $y_{ij} = 1$ and $z_{ij}^s = 0$ since $w_{ij} - \sum_{s=1}^{S} p_s c_{ij}^s < 0$. If y_{ij} is relaxed to $y_{ij} \le 1$, the above problem is also optimized at $y_{ij} = 1$ and $z_{ij}^s = 0$.

Summarizing all cases above, we can obtain that under *Assumption A*, the decision variables $y_{ij}, z_{ij}^s \in \{0,1\}$ in TSGP can be relaxed by $0 \le y_{ij} \le 1, z_{ij}^s \ge 0$. □

For the nonlinear term $x_{ik}^s x_{jk}^s$, by introducing $u_{ijk}^s = x_{ik}^s x_{jk}^s$, we can have following linearization methods.

Lemma 2. *The constraint (6) for edge $(v_i, v_j) \in E$ in scenario s is equivalent to following linear constraints:*

$$\begin{cases} y_{ij} + z_{ij}^s = 1 - \sum_{k=1}^{K} u_{ijk}^s \\ u_{ijk}^s \le x_{ik}^s \\ u_{ijk}^s \le x_{jk}^s \\ u_{ijk}^s \ge x_{ik}^s + x_{jk}^s - 1 \\ u_{ijk}^s \ge 0 \end{cases} \tag{8}$$

By Lemma 1 and Lemma 2, we have the following theorem, which presents the equivalent integer linear programming formulation for TSGP.

Theorem 2. *The formulation in (2)-(7) for TSGP under* Assumption A *is equivalent to the following integer linear program in extensive form:*

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} w_{ij}y_{ij} + \sum_{s=1}^{S} p_s \sum_{(i,j)\in E} c_{ij}^s z_{ij}^s \\ s.t. \quad & (5), (8) \\ & x_{ik}^s \in \{0,1\}, 0 \le y_{ij} \le 1, z_{ij}^s \ge 0 \\ & i \in V, (i,j) \in E, k = 1, \cdots, K, s = 1, \cdots, S \end{aligned}$$

For some specific case, we can have a more simplified formulation for TSGP as shown in the following corollary.

Corollary 2. *If $w_{ij} > 0$ and $c_{ij}^s > 0$ hold for all edge $(v_i, v_j) \in E$ for all scenarios, the formulation in (2)-(7) for TSGP is equivalent to:*

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in E} w_{ij}y_{ij} + \sum_{s=1}^{S} p_s \sum_{(i,j)\in E} c_{ij}^s z_{ij}^s && (9)\\
s.t. \quad & \sum_{k=1}^{K} x_{ik}^s = 1, \sum_{i=1}^{N} x_{ik}^s = n_k && (10)\\
& -(y_{ij} + z_{ij}^s) - x_{ik}^s + x_{jk}^s \le 0 && (11)\\
& -(y_{ij} + z_{ij}^s) + x_{ik}^s - x_{jk}^s \le 0 && (12)\\
& x_{ik}^s \in \{0,1\}, 0 \le y_{ij} \le 1, z_{ij}^s \ge 0 && (13)\\
& i \in V, (i,j) \in E, k = 1, \cdots, K, s = 1, \cdots, S
\end{aligned}
$$

Proof. For edge $(v_i, v_j) \in E$ such that $w_{ij} > 0$ and $c_{ij}^s > 0$ in scenario s, different from Lemma 1, if vertices v_i, v_j are in the same subset, i.e., $x_{ik'}^s = x_{jk'}^s = 1$ and $x_{ik}^s = x_{jk}^s = 0$ for all $k \neq k'$, we have $y_{ij} + z_{ij}^s \ge 0$ with considering all k's in (11) and (12); Similarly, if vertices v_i, v_j are different subsets, we have $y_{ij} + z_{ij}^s \ge 1$ from (11) and (12). As mentioned in Lemma 1, the objective function (9) with respect to edge (v_i, v_j) is $\min(w_{ij}y_{ij} + \sum_{s=1}^S p_s c_{ij}^s z_{ij}^s)$.

For the case $y_{ij} + z_{ij}^s \ge 0$, we want to prove that $y_{ij} = 0$ and $z_{ij} = 0$ by the formulation (9)-(13). we have three subcases:

(a) $w_{ij} > \sum_{s=1}^S p_s c_{ij}^s$. By Corollary 1, $y_{ij} = 0$ and then $z_{ij}^s \ge 0$. Now, $\min(w_{ij}y_{ij} + \sum_{s=1}^S p_s c_{ij}^s z_{ij}^s) = \min \sum_{s=1}^S p_s c_{ij}^s z_{ij}^s$ should obtain the optimal value at $z_{ij}^s = 0$ since all $c_{ij}^s > 0$.

(b) $w_{ij} = \sum_{s=1}^S p_s c_{ij}^s$. By *Assumption A*, $y_{ij} = 0$ and then $z_{ij}^s \ge 0$. This can be proved similarly to above case.

(c) $w_{ij} < \sum_{s=1}^S p_s c_{ij}^s$. We have $y_{ij} + z_{ij}^s \ge 0$. $\min(w_{ij}y_{ij} + \sum_{s=1}^S p_s c_{ij}^s z_{ij}^s)$ should obtain the optimal value at $y_{ij} = 0, z_{ij}^s = 0$ since $w_{ij} > 0$ and $c_{ij}^s > 0$ for all s.

For all subcases, we have $y_{ij} = 0, z_{ij}^s = 0$ when $y_{ij} + z_{ij}^s = 0$, which is the same as the case $y_{ij} + z_{ij}^s = 0$ in Lemma 1.

For the case $y_{ij} + z_{ij}^s \ge 1$, we also discuss three subcases:

(d) $w_{ij} > \sum_{s=1}^S p_s c_{ij}^s$. By Corollary 1, $y_{ij} = 0$ and then $z_{ij}^s \ge 1$. Now, $\min(w_{ij}y_{ij} + \sum_{s=1}^S p_s c_{ij}^s z_{ij}^s) = \min \sum_{s=1}^S p_s c_{ij}^s z_{ij}^s$ should obtain the optimal value at $z_{ij}^s = 1$ since all $c_{ij}^s > 0$.

(e) $w_{ij} = \sum_{s=1}^S p_s c_{ij}^s$. By *Assumption A*, $y_{ij} = 0$ and then $z_{ij}^s \ge 1$. This can be proved similarly to above case.

(f) $w_{ij} < \sum_{s=1}^S p_s c_{ij}^s$. We have $y_{ij} + z_{ij}^s \ge 1$. If $y_{ij} = 0$, $\min(w_{ij}y_{ij} + \sum_{s=1}^S p_s c_{ij}^s z_{ij}^s)$ should obtain the optimal value $\sum_{s=1}^S p_s c_{ij}^s$ at $z_{ij}^s = 1$ for all s since $z_{ij}^s \ge 1$ and $c_{ij}^s > 0$ for all s; If $y_{ij} = 1$, the objective for edge (v_i, v_j) obtain the optimal value w_{ij} at $z_{ij}^s = 0$ for all s since $z_{ij}^s \ge 0$ and $c_{ij}^s > 0$ for all s; If $0 < y_{ij} < 1$, assume the optima for $\min(w_{ij}y_{ij} + \sum_{s=1}^S p_s c_{ij}^s z_{ij}^s)$ is obtained at $y_{ij} = a$ $(0 < a < 1)$, and thus $z_{ij}^s \ge 1 - a$.

Then

$$\min(w_{ij}y_{ij}+\sum_{s=1}^{S} p_s c_{ij}^s z_{ij}^s) = aw_{ij} + \min\sum_{s=1}^{S} p_s c_{ij}^s z_{ij}^s$$

with constraints $z_{ij}^s \geq 1-a$ obtains its optimal value $aw_{ij}+(1-a)p_s c_{ij}^s$ at $z_{ij}^s = 1-a$ for all s. Comparing the objective values $\sum_{s=1}^{S} p_s c_{ij}^s, aw_{ij}+(1-a)p_s c_{ij}^s, w_{ij}$ at $y_{ij}=0,a,1$, respectively, we have

$$\sum_{s=1}^{S} p_s c_{ij}^s > aw_{ij}+(1-a)p_s c_{ij}^s > w_{ij}$$

for any $0<a<1$, by considering $w_{ij} < \sum_{s=1}^{S} p_s c_{ij}^s$. Therefore, under this subcase, the optimal value is obtained at $y_{ij}=1$ and $z_{ij}^s = 0$ for all s.

The solutions for the case $y_{ij}+z_{ij}^s \geq 1$ under certain situations are the same as those in Lemma 1 when $y_{ij}+z_{ij}^s = 1$. □

In Corollary 2, we require that for any edge $(v_i,v_j)\in E$. $w_{ij}>0$ and $c_{ij}^s>0$ for all s. For example, in case (d), if $c_{ij}^s=0$, the term $\min c_{ij}^s z_{ij}^s$ with $z_{ij}\geq 1$ can obtain the minimum value 0 at any value of nonnegative z_{ij}^s; in case (c), $w_{ij}=0$ will influence any nonnegative choice of y_{ij}.

Comparing formulations in Theorem 2 and Corollary 2, we can see that the formulation in Corollary 2 reduces a lot of constraints and variables, and it can be solved more efficiently. Our method for solving TSGP is using CPLEX to solve the binary linear programs in extensive form in Theorem 2 and Corollary 2.

4 Experimental Results

In this section, we implement all binary linear programs in Theorem 2 and Corollary 2 using CPLEX 11.0 via ILOG Concert Technology 2.5. All computations are performed on a SUN UltraSpace-III with a 900 MHz processor and 2.0 GB RAM. Computational times are reported in CPU seconds.

To test our algorithms, we generate random graphs with different number of nodes, edges and scenarios. The number of edges is determined by the density r of a graph, which is the ratio of the number of edges and the number of possible edges. All generated weights satisfy $w_{ij}, c_{ij}^s \in [2,3]$ for $(v_i,v_j)\in E$. The number $N(N-1)r/2$ of decision variables y_{ij}s is related to the number of edges, which is related to N,r; the number $N(N-1)rS/2$ of z_{ij}^ss is determined by N,r,s; and the number $N\cdot K\cdot S$ of x_{ik}^ss is determined N,S,K. In Theorem 2, the variables u_{ijk}^ss are introduced, and the number of u_{ijk}^ss is $N(N-1)rKS/2$.

In Table 1, the objective values obtained by solving formulations in Theorem 2 and Corollary 2 are the same while the gap in CPLEX is setting to 1%.

From Table 1, we can see all computational seconds by the formulation in Corollary 2 are less than or equal to seconds by the formulation in Theorem 2 under the same test cases. As discussed in Section 3, the formulation in Theorem 2 introduces the variables u_{ijk}^ss and this influences the computational complexity. For the graph with $N=10$ vertices and $S=2$ scenarios, when the density r is increasing, the computational seconds

Table 1. Computational results

Graphs			Probability Dist.	Cardinalities	Objective Value	CPU Seconds	
N	r	S	$p_1+\cdots+p_S$	$n_1+\cdots+n_K$	Thm 2/Cor 2	Thm 2	Cor 2
10	0.1	2	0.6+0.4	5+5	0.00	0.01	0.01
	0.4				13.49	0.10	0.06
	0.7				31.61	0.50	0.33
	1				57.91	17.30	15.84
	0.1	2	0.6+0.4	3+3+4	2.11	0.04	0.03
	0.4				19.72	3.20	2.11
	0.7				43.94	23.04	17.78
	1				76.77	2295.31	1828.23
	0.1	2	0.6+0.4	3+2+3+2	2.11	0.05	0.04
	0.4				22.18	4.95	4.04
	0.7				52.08	170.38	120.70
	1				86.14	5088.86	3583.21
10	0.1	4	0.1+0.2+0.3+0.4	4+3+3	0.00	0.03	0.03
	0.4				20.80	90.62	64.15
	0.1	6	0.05+0.10+0.15+0.20+0.25+0.25	3+3+4	2.21	1.33	0.86
	0.4				19.47	1938.17	774.08
15	0.1	2	0.6+0.4	5+5+5	4.58	0.09	0.08
	0.4				43.42	42.33	23.41
20	0.1	2	0.6+0.4	7+7+6	6.52	0.85	0.42
	0.4				85.22	4354.52	2519.88
25	0.1	2	0.6+0.4	8+8+9	13.95	7.00	1.23
30	0.1	2	0.6+0.4	10+10+10	22.33	11.23	1.44

are increasing as well; for the graph $N = 10, S = 2$ and the same density r, when the number K of subsets increases, the computational seconds increase as well for both formulations. Similarly, when the number S of scenarios and the number N of vertices are increasing, computational seconds increase as well.

5 Conclusions

In this paper, we present the two-stage stochastic graph partitioning problem by the stochastic programming approach. This problem is formulated as a nonlinear stochastic mixed integer program, and we also present a linear programming (SMIP) approach for solving this problem by CPLEX. For some cases with specific requirements, we present a more simplified linearization method, which can solve the problem more efficiently. The SMIP problem is very hard to solve, and for further research, more efficient algorithms, such as methods based on Benders decomposition [1], should be designed for solving large-scale two-stage stochastic graph partitioning problems.

References

1. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik 4, 238–252 (1962)
2. Birge, J.R., Louveaux, F.: Introduction to stochastic programming. Springer, Heidelberg (1997)
3. Fan, N., Pardalos, P.M.: Linear and quadratic programming approaches for the general graph partitioning problem. Journal of Global Optimization 48(1), 57–71 (2010)
4. Fan, N., Pardalos, P.M.: Multi-way clustering and biclustering by the Ratio cut and Normalized cut in graphs. Journal of Combinatorial Optimization (2010), doi:10.1007/s10878-010-9351-5
5. Fan, N., Pardalos, P.M.: Robust optimization of graph partitioning and critical node detection in analyzing networks. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part I. LNCS, vol. 6508, pp. 170–183. Springer, Heidelberg (2010)
6. Frieze, A., Flaxman, A., Krivelevich, M.: On the random 2-stage minimum spanning tree. Random Structures and Algorithms 28, 24–36 (2006)
7. Kong, N., Schaefer, A.J.: A factor 1/2 approximation algorithm for two-stage stochastic matching problems. European Journal of Operational Research 172, 740–746 (2006)

A Spatio-Temporal Approach to the Discovery of Online Social Trends

Harshavardhan Achrekar, Zheng Fang, You Li, Cindy Chen, Benyuan Liu, and Jie Wang

Department of Computer Science
University of Massachusetts, Lowell, MA 01854, USA

Abstract. Online social networks (OSNs) have become popular platforms for people to interact with each other in the cyber space. Users use OSNs to talk about their daily activities, mood, health status, sports events, travel experiences, political campaigns, entertainment events, and commercial products, among other things. Conversations between users on an OSN site could reflect the current social trends that are of great interest and importance for individuals, businesses, and government agencies alike. In this paper we design and develop a comprehensive system to collect, store, query, and analyze OSN data for effective discovery of online social trends. Our system consists of three parts: (1) an OSN data collection engine; (2) a spatio-temporal database for storing, indexing, and querying data; and (3) a set of analytical tools for online social trend discovery. We demonstrate the effectiveness of our system using a recent result of predicting seasonal flu trends using Twitter data.

1 Introduction

Online social networking (OSN) web sites, such as Facebook, MySpace, and Twitter, have become popular platforms for people to interact with each other and form cyber communities. The OSN sites provide functionalities for users to build online profiles, establish and maintain friendships, send messages, post photos, advertise events, comment on profiles of friends, and create and join online communities of common interests.

The rapid growing of the number of OSN users and services has made online social networking a vital part of the Internet ecosystem. Major OSN sites today have more than half a billion users worldwide. At any given time, tens of millions of users are logged in various OSN sites, where each user on average would spend tens of minutes daily on the sites. Each week billions of blog articles, web links, and photos are posted and shared between friends.

For a large number of users, going to OSNs has become their daily activities. They use OSNs to talk about their daily activities, mood, health status, sports events, travel experiences, political campaigns, entertainment events, commercial products, to name just a few. We note that conversations between users on the OSN sites could reflect the current social trends that are of interests to certain groups of individuals, businesses, and government agencies. For example, fashion industry may follow the OSN discussions among targeted customers to keep track of the fashion trends for designing and marketing their fashion products. Politicians may use OSNs to gauge public opinions on various issues to better understand and serve the needs of their constituencies, or direct their election campaigns. Technology companies may use OSNs to obtain user feedbacks when rolling out new products. Moreover, OSNs may also be used to monitor public health trends, including the outbreak of seasonal influenza or the H1N1 Swine flu [1].

Online social trends may change with time, geographical locations, and demographics of the users such as age ranges and gender. We would like to develop methodology and tools to automatically discover online social trends. Such tools are expected to have significant impacts on societal, economic, and political issues. Realizing the importance of identifying social trends, certain OSN sites, e.g., Twitter Trending Topics, have started to provide rudimentary reports of online social trends. These reports, unfortunately, often just provide limited relevant information and functions, or without sufficient details to help discover social trends. We tackle this problem in this paper. In particular, we design and develop a comprehensive framework to collect, store, query, and analyze OSN data for effective discovery of online social trends. The architecture of our system is depicted in Figure 1. The functions and inter-relationships of the main components are briefly described below.

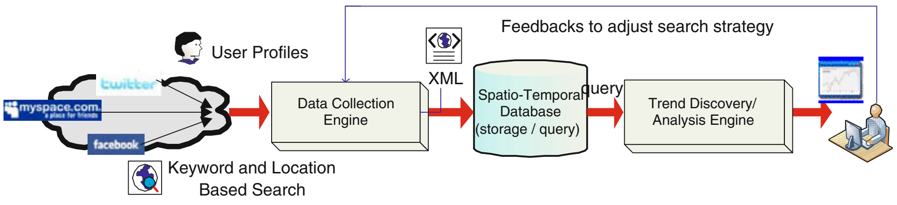

Feedbacks to adjust search strategy
User Profiles
twitter
myspace.com
facebook
Data Collection Engine
XML
Spatio-Temporal Database (storage / query)
query
Trend Discovery/ Analysis Engine
Keyword and Location Based Search

Fig. 1. System architecture for online social trends discovery

1. *Online Social Network Data Collection.* The data collection engine is used to search and retrieve data from OSN sites. Given a set of keywords such as "flu", "health care reform", and "Windows 7", the engine continuously crawls the OSN sites and collects relevant blogs (content and time stamp) and user profile information when available including age, gender, and location, among other things. The collected information flow, i.e., a stream of high-dimensional spatio-temporal data, is aggregated and fed to a spatio-temporal database for storing, indexing, and querying.
2. *Data Storing, Indexing, and Querying.* The data collection engine would produce a large volume of streaming data with spatio-temporal aspects. Efficient data storage and indexing mechanisms and effective querying platforms are thus needed. We design and develop a database system called `OSNDB` to store

the collected data. A novel "partitioning and labeling" based analytical indexing method is devised to speed up the query process for high-dimensional data. Furthermore, we create a flexible and efficient query platform to support complex spatio-temporal stream queries for these data.

3. *Online Social Trends Discovery.* The data storage and indexing scheme allows for efficient queries on combinations of fields including age range, gender, and location. We further provide proper analytical tools to help discover online social trends. As a case study, we collect and analyze the seasonal flu trends from major OSN sites and compare our results with the CDC flu data [1]. The analysis will also provide important feedbacks to the data collection engine to adjust its collection strategy.

The rest of the paper is structured as follows. In Section 2 we describe our OSN data collection method. In Section 3 we present our spatio-temporal database system designed for online social trend discovery. As a case study, we describe our recent result on predicting seasonal flu trends using Twitter data in Section 4. Finally, we summary our work in Section 5.

2 Online Social Networks Data Collection

We present a methodology to collect relevant information from Facebook, MySpace, and Twitter, as well as other major OSN sites. These three OSN sites constitute a large fraction of the total number of OSN users. Each of these three sites has its own focus and appeals to different groups of users. In particular, MySpace is especially popular among teenagers, musicians, artists, and other entertainment related entities. Facebook, started as a social networking site for college students, has since percolated to other users groups including high school students, graduate students, and professionals. Twitter is mainly used as a real-time micro-blogging system for people and media to distribute and share topics of their interests.

2.1 Design of OSN Data Collection Engines

Facebook, MySpace, and Twitter each provides a search interface (API) that allows a user to enter any keywords to search for blogs that contain these keywords. The APIs typically return a list of blog entries across the entire site that contains the keywords in the reverse-time order. Each entry contains publisher profile ID, time stamp of the blog, and the corresponding blog content. The individual fields can be obtained using an HTML content scraper. Given a profile ID, we can retrieve the publisher profile information from the OSN sites, which typically includes the name, age, gender, geographic location, friends/followers, and other information of the user.

We devise and implement an OSN crawler for each of the three OSN sites based on the search APIs provided by these OSN sites. In what follows we will describe the design of Facebook crawler in details. The architecture and the main components of the crawlers for MySpace and Twitter are similar, albeit different

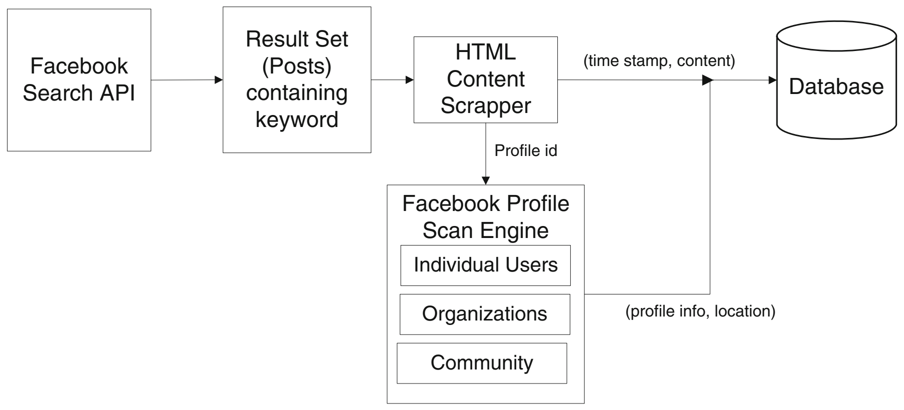


Fig. 2. Design of the Facebook data collection engine

search APIs, returned result formats, privacy settings, and the underlying web technologies.

The structure of the Facebook crawler is depicted in Figure 2. The function of each component is described below.

1. *Facebook Search Engine*: After signing in Facebook with a valid account, we can enter keywords and search for updates and posts that contain the keywords. The "Post by everyone" option allows us to search for given keywords in the updates and posts of all users on Facebook.
2. *Results Set containing keyword*: Users are given a few options to post updates on Facebook. The common options are "friends", "group", and "everyone". The "everyone" option is the default setting, which makes the corresponding updates available to the public and the Facebook search engine. Results returned by the search engine are available to the public for a limited period of time.
3. *HTML Content Scrapper*: The HTML content scrapper is a screen scrapper for web pages. We are interested in getting useful information out of posts returned from the keyword search. The HTML content is send as input onto a regular expression matcher. Techniques of pattern matching are applied to extract relevant content. We are interested in the following three fields: profile ID, time stamp of the post, and the post content.
4. *Facebook Profile Scan Engine*: Given a profile ID, we will retrieve the detailed information of the profile, which typically includes, among other things, name, gender, age, affiliations (school, work, region), birthday, location, education history, and friends. Note that a profile may belong to an individual user, an organization, or a community.

The information collected in this process will be aggregated and fed to the database for storing, indexing, and querying. The most important fields of each record include the profile information (name, gender, age, etc), location, time stamp, and the blog content.

3 The Spatio-Temporal Database System

We design and develop a database system called OSNDB to facilitate efficient data storage and effective information extraction specific to online social networks. Data from online social networks differs from conventional data in the following two aspects:

1. OSN data is streamlined [17] and thus needs to be stored in real time where moving-window queries [21] are necessary.
2. The spatio-temporal aspect of the data [8,7] is crucial, i.e., people will want to see when and where most messages related to a certain topic have been posted.

To address these two issues, we design and develop the Online Social Networks DataBase system (OSNDB). In the following sections, we will describe the database design, access methods, and query platform. The design of the database system is depicted in Figure 3.

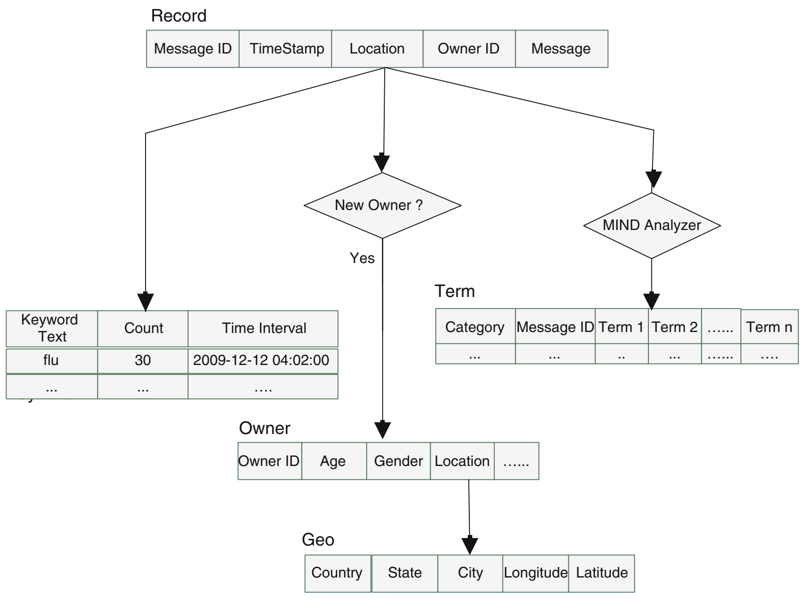


Fig. 3. Design of the Online Social Networks Database System (OSNDB)

3.1 Database Design

Once the data is collected, each entry will be stored into the `Record` table instantaneously. The `Record` table contains five fields: `Message ID, Timestamp, Location, Owner, Message`. `Message ID` is automatically generated by the database system. `Timestamp` records the time when the entry is posted. `Owner` is the identifier of the person who has posted the message. If the owner has indicated his/her location, the `Location` field will remember this information.

The `OSNDB` database system also maintains a `Keyword` table, which contains three fields: `Keyword Text, Count, and Time Interval`. In this table, terms that appear most frequently in the past hour will be recorded. While the default granularity is one hour, other granularity can be generated by aggregate operations.

Posted messages will then be processed. We will use a natural language analyzing system (e.g., the MIND framework [6]) to interpret the messages. MIND is designed to handle abbreviated or imprecise user inputs in human-computer conversations. It uses a variety of contexts, such as domain context and conversation context to reach a full understanding of user inputs, including those abbreviated or imprecise ones.

The result of MIND analyzer will be stored in the `Term` table. The `Term` table contains the message ID, category of the message, the meanings associated with the message. For example, let us consider the following message posted on Facebook: "*I still don't want the shot. I have had swine flu since Oct. I was at home for two weeks. I have been working pretty much full time since, cut keep feeling tired, getting a cough, sore throat. If I take grapefruit seed extract I am okay = when I stop the symptoms come back. I want it to END! This is ridiculous.*". This message will be processed by MIND, and the important terms in this message – "swine flu", "shot", "tired", "cough", "sore throat", "grapefruit seed extract" will be extracted and stored in the `Term` table and the message will be categorized as "swine flu".

`OSNDB` also contains an `Owner` table and a `Friend` table. The `Owner` table record the information about the owners of the messages, such as `Owner ID, age, gender, location`. The `Friend` table will record friendship relation between owners.

In addition, `OSNDB` has a `Geo` table to record geographical information, including `Country`, `State`, `City`, `Longitude`, and `Latitude`.

3.2 Access Methods

To speed up the querying process, we create the following standard indices on these tables:

1. A B+-tree [2] index on the `Timestamp` field of the `Record` table.
2. A R*-tree [3] index on the `Location` field of the `Record` table.
3. A B+-tree index on the `Time Interval` field of the `Keyword` table.

The `Term` table contains more than 10 fields, which is high-dimensional. To index data records in the `Term` table, we have explored existing high-dimensional indexing methods. Some of them do not scale well with dimensionality, for example, the R-tree family [15,23,3]; some are too complicated to implement, for example, the Pyramid technique [4] and the *iMinMax* method [25]. The X-tree [5] is currently the dominating high-dimensional indexing method, which organizes the directory using a splitting algorithm to minimize overlaps with the mechanism of super-nodes. The basic idea of overlap-minimizing split and super-nodes is to keep the directory in a hierarchy as much as possible, which helps to prevent splits and avoid high overlaps. However, it is difficult to determine the maximum overlap, for an excessive number of super-nodes would decrease the selectivity. X-trees are more suited for medium-dimensional data space.

In a previous study, we have developed a new indexing technique called ***PL-tree***, a short-hand for “partitioning and labeling” [24]. Different from existing high-dimensional algorithms which are all geometry based, the PL-tree is an analytical indexing method. The PL-tree uses a pairing function to generate a unique label for a group of data that are geographically close to each other. The main idea of PL-tree is as follows:

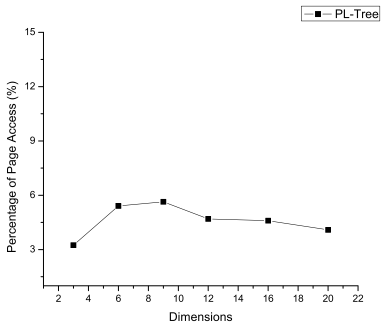
1. Partition the original space into hypercubes, where each hypercube is called a sub-home.
2. Map objects in a sub-home to a fixed point that is unique to the sub-home.
3. Label each object in a sub-home using a pairing function on the fixed point so that all objects in the same sub-home have the same label and objects in different sub-homes have different labels.
4. If the number of objects contained in a sub-home is greater than a predetermined bound (e.g. the page size of the underlying operating system), continue this process recursively on each sub-home.

The structure of a PL-tree is a tree of labels with an unbounded number of children at each node, where each label uniquely identifies the set of objects contained in the corresponding sub-home of the label. The PL-tree indexing cuts down search redundancy in range queries, and is especially suitable for indexing large volumes of high-dimensional data. The uniqueness property of labeling plays an important role in making range queries efficient.

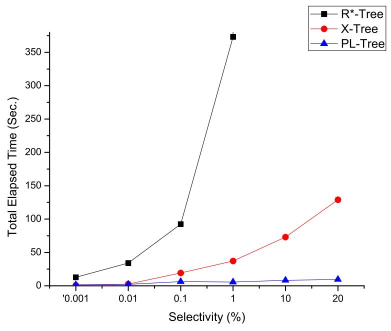
Figure 4 shows our preliminary performance results. It can be seen that the PL-tree indexing maintains a rather smooth performance regardless the increase of dimensionality in terms of percentage of page access. Also, for the total elapsed time, the PL-tree indexing significantly outperforms the X-tree [5], the currently dominating high-dimensional indexing method.

3.3 Query Platform

We use user-defined aggregates and user-defined functions to implement special operators needed in the `OSNDB` system. User-defined aggregates can be created in advanced database management software, e.g., Oracle 11g. The aggregates are applied to the data stored at the physical level. Our query language has the following two properties:



(a)



(b)

Fig. 4. (a) Page access percentage over data set dimensions (b) Performance over real multidimensional data sets

1. *Ease of Use:* Queries in the `OSNDB` system conform to the SQL:2003 standard [9]. No new constructs needs to be introduced.
2. *Extensibility:* In addition to pre-defined special operators, users can define their operators as user-defined aggregates.

To illustrate the ease of use and extensibility of the query platform of the `OSNDB` system, we provide the following sample queries using OSN flu trend as an example.

Query 1: Find when the term “swine flu” was most frequently recorded.

```
SELECT time_interval
FROM keyword
WHERE keyword_term = ‘‘Swine Flu’’
HAVING (SELECT count(keyword_term) FROM keyword
            WHERE keyword_term = ‘‘Swine Flu’’
        >= ALL (SELECT count(keyword\_term) FROM keyword))
```

Query 2: Find when the term “swine flu” appears together with the term “flu shot” .

```
SELECT intersect(k1.time_interval, k2.time_interval)
FROM keyword AS k1, keyword AS k2
WHERE k1.keyword_term = ‘‘Swine Flu’’ AND k2. keyword_term= ‘‘flu shot’’
```

In Query 2, `intersect` is an user-defined temporal aggregate which calculates the time period when the time intervals associated with swine flu and the time intervals associated with flu shot intersects.

Query 3: Find the average frequency of the term “swine flu” for every 100 new values inserted into the `Keyword` table.

```
SELECT moving_average (count, 100)
FROM keyword
WHERE keyword_term = ‘‘Swine Flu’’
```

Query 4: Find where people are talking about “swine flu” most frequently.

```
SELECT city, state
FROM record, term, geo
WHERE terms LIKE ‘‘swine flu’’ AND terms.id = record.id
GROUP BY city, state
HAVING count(*) >= ALL (SELECT count(*)
                        FROM record, term, geo
                        WHERE terms LIKE ‘‘swine flu’’
                        AND terms.id = record.id
                        GROUP BY city, state)
```

In OSNDB, we use user-defined aggregates and user-defined functions to extend the query capability of the standard SQL without introducing new constructs. For example, the user-defined aggregate MOVING_AVERAGE in Query 3 is implemented as follows in a previous publication [7].

4 Predicting Flu Trends Using Twitter Data: A Case Study

In this section, as a case study we describe our recent result of predicting flu trends using Twitter data [1].

Seasonal influenza epidemics result in about three to five million cases of severe illness, which cause about 250,000 to 500,000 deaths worldwide each year [16]. Reducing the impact of seasonal epidemics and pandemics such as the H1N1 influenza is of paramount importance for public health authorities. Studies have shown that preventive measures can be taken to contain epidemics, if an early detection is made during the germination of an epidemic [11,20]. Therefore, it is important to track and predict the emergence and spread of flu in the population.

The Center for Disease Control and Prevention (CDC) monitors influenza-like illness (ILI) cases, by collecting data from sentinel medical practices, collating the reports and publishing them on a weekly basis. As diagnoses are made and reported by doctors, the system is almost entirely manual, resulting in a 1-2 weeks delay between the time a patient is diagnosed and the moment that data point becomes available in aggregate ILI reports. Public health authorities need to be forewarned at the earliest time to ensure effective preventive intervention, and this leads to the critical need of more efficient and timely methods of estimating influenza incidences.

Several innovative surveillance systems have been proposed to capture the health seeking behavior and transform them into influenza activity indicators. These include monitoring call volumes to telephone triage advice lines [10], over the counter drug sales [22], patients visit logs to Physicians for flu shots, and Google Flu Trends which utilizes aggregated web search queries pertaining to influenza to build a comprehensive model that can estimate nationwide as well as state-level ILI activity [13].

In this work we investigate the use of a novel data source, namely, messages posted on Twitter, to track and predict the level of ILI activity in a population. Data collected from twitter represents a previously untapped data source for detecting the onset of a flu epidemic and predicting its spread. Our approach assumes twitter users as "sensors" and the collective message exchanges with a mention of flu such as *"I got Flu"* and *"down with swine flu"* as early indicators and robust predictors of influenza. Although many of these data are noisy individually, in aggregate they reveal the underlying epidemic pattern in time and space.

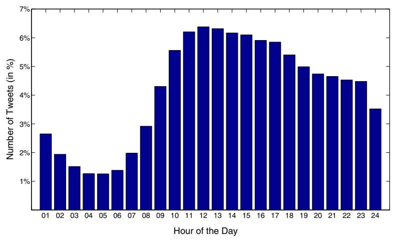
4.1 Data Sets

We searched and collected tweets and profile details of Twitter users who mentioned about flu descriptors in their tweets starting from October 18th, 2009 and lasting until October 31st, 2010. There are 4.7 million tweets from 1.5 million unique users. Location details can be set to public or private from the profile page or mobile client. In our Twitter dataset, 30.6% users were from USA, 41.3% users were from outside USA, and 28.1% users did published their location details. Within USA, we have seen users who tweeted about flu in all 50 states and the District of Columbia.

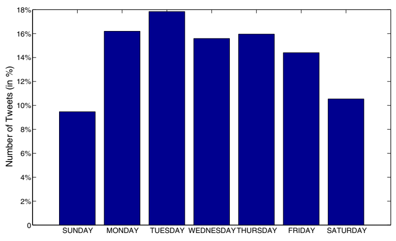
Figure 5(a) shows the percentage of unique Twitter users who mentioned about flu in tweets at different hours of the day. Status posting times (tweet timestamp in GMT) are converted to the local timezone of the individual profile. Day light saving time is also applied within the required time frame. The hourly activity patterns observed at different hours of the day are what we expected, with high traffic volumes from late morning to early afternoon and less tweet posted from midnight to early morning, reflecting people's work and rest hours within a day. The average daily usage pattern within a week shown in Figure 5(b) suggests a trend on OSN sites with more people discussing about flu on weekdays than on weekends. Note that our observed usage patterns in Twitter are consistent with previous observations for other OSNs [12].

For analysis, Twitter dataset needs to be processed to discount retweets and successive posts from the same users within a certain syndrome elapsed time. These two issues are explained below.

1. *Retweets*: A retweet is a post originally made by one user that is forwarded by another user. For flu tracking, a retweet does not indicate a new ILI case, and thus should not be counted in the analysis. Out of 4.7 million tweets we collected, there are 450,000 retweets, accounting for 9.5% of the total number of tweets.
2. *Syndrome elapsed time*: An individual patient may have multiple encounters associated with a single episode of illness (e.g., initial consultation, consultation 1-2 days later for laboratory results, and followup consultation a few weeks later). To avoid duplication from this common pattern of ambulatory



(a)



(b)

Fig. 5. (a) Hourly Twitter usage pattern in USA (b) Average daily Twitter usage within a week

care, the first encounter for each patient within any single syndrome group is reported to CDC, but subsequent encounters with the same syndrome are not reported as new episodes until six weeks has elapsed since the most recent encounter in the same syndrome [18]. We call it syndrome elapsed time. In our dataset, we remove tweets from the same user within a certain syndrome elapsed time, since they do not indicate new ILI cases.

We created different datasets consisting of the original Twitter dataset, Twitter dataset without retweets, Twitter dataset without retweets and having no tweets from the same user within a syndrome elapsed time of 1 week, 2 weeks, and 6 weeks, respectively. Comparing these different dataset with CDC data we found that the dataset without retweets and having no tweets from the same user within syndrome elapsed time of one week yields the highest correlation coefficient (0.9846) with the CDC data. This dataset will be used for all successive experiments. The relationship between the dataset and the CDC data is illustrated in Figure 6, which shows a very close to linear relationship between the two datasets.

4.2 Twitter Improves Prediction of Influenza Data

We use statistical models to predict the number of Influenza Like Illness (ILI) incidents in the following week. The models use the data collected and published by the CDC, as the percentage of visits to "sentinel" physicians attributable to ILI in successive weeks. We test our models using the CDC data collected in the previous period, with and without measures of Twitter activities. We show that Twitter data improves accuracy substantially.

Model Structure. We use the following form of auto-regression with exogenous inputs (ARX) [19,14] in our experiments:

$$y(t) = \sum_{i=1}^{m} a_i y(t-i) + \sum_{j=0}^{n-1} b_j u(t-j) + c + e(t) \tag{1}$$

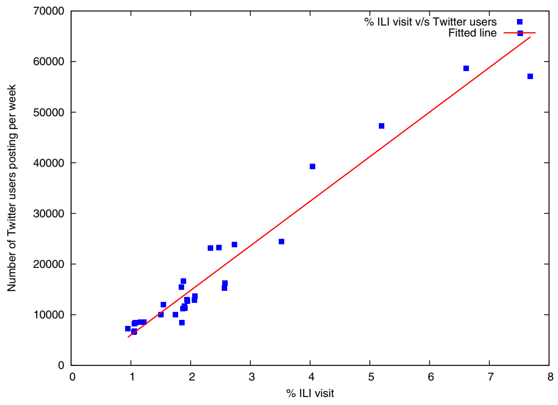


Fig. 6. Number of Twitter users per week versus percentage of weighted ILI visit by CDC

where t indexes weeks, $y(t)$ denotes the percentage of physician visits for ILI in week t, $u(t)$ represents the number of unique Twitter users with flu related tweets in week t, $e(t)$ is a sequence of independent random variables, and c is a constant to account for offset.

We note that Twitter data provides some independent real-time assessment of influenza as events in Twitter text, and Eq. 1 takes this into consideration. Unfortunately, data is carried by a chaotic, noisy data stream and may be disturbed from time to time by events not directly related to cases of ILI. The CDC aims to provide, with delays, a valid physician confirmed ILI diagnosis to measure "true" population ILI activities.

In our experiments, we vary m from 0 to 2 and n from 0 to 3 in Eq. 1. Within these ranges, $m = 0$ or $n = 0$ represent the models where there are no CDC data y or Twitter data u terms present respectively. Also, if $m = 0$ and $n = 1$, we have a linear regression between the Twitter data and the CDC data. If $n = 0$, we have standard auto-regressive (AR) models. The AR models utilize past CDC data, and serve as baselines to validate whether Twitter data provides predictive power beyond historical CDC data.

The objective of the model is to provide timely updates of the percentage of physician visits. To predict such percentage in week t, we assume that only the CDC data with at least 2 weeks of lag is available for the prediction if past CDC data is present in a model. For the Twitter data, we assume that the most recent data is always available if a model includes the Twitter data terms.

To predict the flu cases in week t using the ARX model in Eq. 1 based on the CDC data with 2 weeks of delay and/or the up-to-date Twitter data, we apply the following relationships:

$$\hat{y}(t) = a_i\hat{y}(t-1) + \sum_{i=2}^{m} a_i y(t-i) + \sum_{j=0}^{n-1} b_j u(t-j) \tag{2}$$

$$\hat{y}(t-1) = \sum_{i=1}^{m} a_i y(t-i-1) + \sum_{j=0}^{n-1} b_j u(t-j-1) \tag{3}$$

where $\hat{y}(t)$ represents the predicted CDC data in week t. It can be verified from the above equations that to predict the CDC data in week t, the most recent CDC data is from week $t-2$. If the CDC data lag is more or less than two weeks, the above equations can be easily adjusted accordingly.

Cross Validation Results. According to the 5-fold cross validation results, the model corresponding to $m = 0$ and $n = 3$ has the lowest root mean square error (RMSE). The corresponding model has the following form:

$$y(t) = b_0 u(t) + b_1 u(t-1) + b_2 u(t-2) + c. \tag{4}$$

In general, the addition of Twitter data improves the prediction with past CDC data alone. We plot the Twitter dataset (the number of tweets normalized to the same scale as CDC data) and predicted values of the percentage of weighted ILI visits against CDC data in Figure 7 to illustrate the effectiveness of the prediction model.

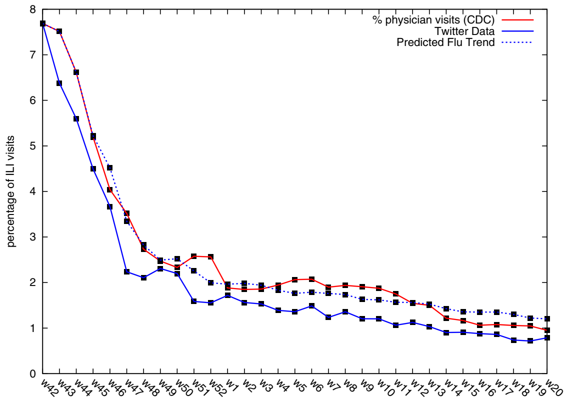


Fig. 7. Twitter dataset normalized to the same scale as CDC data along with its predicted values for percentage of weighted ILI visits(5-fold cross validation)

We tested our regression models with historic CDC data and verified that Twitter data does substantially improve the model accuracy in predicting ILI cases. In view of the lag inherent in CDC's ILI reports, Twitter data provides near realtime assessment of influenza activity and can be used to effectively predict current ILI activity levels.

5 Summary

We design and develop a comprehensive system to collect, store, query, and analyze OSN data for effective discovery of online social trends. Our system consists of an OSN data collection engine, a spatio-temporal database for storing, indexing, and querying data, and analytical tools for discovering online social trends. As a case study, we collect and analyze seasonal flu trends on major OSN sites and compare our results with the CDC flu data. The results show that our system can be used for effective online social trend discovery.

References

1. Achrekar, H., Gandhe, A., Lazarus, R., Yu, S.H., Liu, B.: Predicting flu trends using twitter data. In: International Workshop on Cyber-Physical Networking Systems (CPNS) in conjunction with IEEE Infocom (2011)
2. Bayer, R., McCreight, E.M.: Organization and maintenance of large ordered indices. Acta Inf. 1, 173–189 (1972)
3. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: An efficient and robust access method for points and rectangles. In: Garcia-Molina, H., Jagadish, H.V. (eds.) Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, pp. 322–331. ACM Press, New York (1990)
4. Berchtold, S., Böhm, C., Kriegel, H.P.: The pyramid-technique: Towards breaking the curse of dimensionality. In: SIGMOD Conference. pp. 142–153 (1998)
5. Berchtold, S., Keim, D.A., Kriegel, H.P.: The x-tree: An index structure for high-dimensional data. In: VLDB. pp. 28–39 (1996)
6. Chai, J., Pan, S., Zho, M.: MIND: A Context-based Multimodal Interpretation Framework, Conversational Systems, Natural, Intelligent and Effective Interaction in Multimodal Dialogue Systems. Kluwer Academic Publishers, Dordrecht (2005)
7. Chen, C.X., Wang, H., Zaniolo, C.: Toward extensible spatio-temporal databases: an approach based on user-defined aggregates. In: de Caluwe, R., de Tré, G., Bordogna, G. (eds.) Spatio-Temporal Databases, Flexible Querying and Reasoning, pp. 55–74. Springer, Heidelberg (2004)
8. Chen, C.X., Zaniolo, C.: SQL^T: A spatio-temporal data model and query language. In: Laender, A.H.F., Liddle, S.W., Storey, V.C. (eds.) ER 2000. LNCS, vol. 1920, pp. 96–111. Springer, Heidelberg (2000)
9. Eisenberg, A., Melton, J., Kulkarni, K.G., Michels, J.E., Zemke, F.: SQL:2003 has been published. SIGMOD Record 33(1), 119–126 (2004)
10. Espino, J., Hogan, W., Wagner, M.: Telephone triage: A timely data source for surveillance of influenza-like diseases. In: AMIA: Annual Symposium Proceedings (2003)
11. Ferguson, N.M., Cummings, D.A., Cauchemez, S., Fraser, C., Riley, S., Meeyai, A., Iamsirithaworn, S., Burke, D.S.: Strategies for containing an emerging influenza pandemic in southeast asia. Nature 437 (2005)
12. Gauvin, W., Ribeiro, B., Towsley, D., Liu, B., Wang, J.: Measurement and gender-specific analysis of user publishing characteristics on myspace. IEEE Networks (September 2010)
13. Ginsberg, J., Mohebbi, M.H., Patel, R.S., Brammer, L., Smolinski, M.S., Brilliant, L.: Detecting influenza epidemics using search engine query data. Nature 457 (2009)

14. Goodwin, G.C., Sin, K.S.: Adaptive Filtering Prediction and Control. Prentice-Hall, Inc., Englewood Cliffs (1984)
15. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: SIGMOD Conference, pp. 47–57 (1984)
16. Jordans, F.: WHO working on formulas to model swine flu spread (2009)
17. Koudas, N.: Stream data management: Research directions and opportunities. In: IDEAS (2002)
18. Lazarus, R., Kleinman, K., Dashevsky, I., Adams, C., Kludt, P., DeMaria, Jr., A. R.: Platt: Use of automated ambulatory-care encounter records for detection of acute illness clusters, including potential bioterrorism events (2002)
19. Ljung, L.: System Identification: Theory for the User. Prentice-Hall, Inc., Upper Saddle River (1999)
20. Longini, I., Nizam, A., Xu, S., Ungchusak, K., Hanshaoworakul, W., Cummings, D., Halloran, M.: Containing pandemic influenza at the source. Science 309(5737) (2005)
21. Qiao, L., Agrawal, D., Abbadi, A.E.: Supporting sliding window queries for continuous data streams. In: SSDBM (2003)
22. Magruder. S.: Evaluation of over-the-counter pharmaceutical sales as a possible early warning indicator of human disease. Johns Hopkins University APL Technical Digest (2003)
23. Sellis, T.K., Roussopoulos, N., Faloutsos, C.: The R+-tree: A dynamic index for multi-dimensional objects. In: Stocker, P.M., Kent, W., Hammersley, P. (eds.) Proceedings of 13th International Conference on Very Large Data Bases, VLDB 1987, Brighton, England, September 1-4, pp. 507–518. Morgan Kaufmann, San Francisco (1987)
24. Wang, J., Fang, Z., Chen, C.X.: The pl-tree: A fast high-dimensional access method for range queries. Technical Report, Department of Computer Science, University of Massachusetts Lowell (2009)
25. Yu, C., Ooi, B.C., Tan, K.L., Jagadish, H.V.: Indexing the distance: An efficient method to knn processing. In: VLDB. pp. 421–430 (2001)

A New Approximation Algorithm for the Selective Single-Sink Buy-at-Bulk Problem in Network Design

Peng Zhang*

School of Computer Science and Technology, Shandong University,
Jinan 250101, China
algzhang@sdu.edu.cn

Abstract. The Selective Single-Sink Buy-at-Bulk problem was proposed by Awerbuch and Azar (FOCS 1997). For a long time, the only known non-trivial approach to approximate this problem is the tree-embedding method initiated by Bartal (FOCS 1996). In this paper, we give a thoroughly different approximation approach for the problem with approximation ratio $O(\sqrt{q})$, where q is the number of source terminals in the problem instance. Our approach is based on a mixed strategy of LP-rounding and the greedy method. When the number q (which is always at most n) is relatively small (say, $q = o(\log^2 n)$), our approximation ratio $O(\sqrt{q})$ is better than the currently known best ratio $O(\log n)$, where n is the number of vertices in the input graph.

1 Introduction

The Single-Sink Buy-at-Bulk Network Design problem arises in the following scenario in practice. We are asked to design a capacitated network to route flows from a given set of source terminals to a specified sink terminal. For example, the sources may be oil wells managed by an oil company and the sink may be a major refinery, as shown in [17]. We can design the capacitated network by installing on edges links from an available set of link types, with each type of link having a capacity and a price per unit of length. The prices and capacities of link types have economy of scale so that the price-capacity ratio is smaller for link types with higher capacity. The installed links must satisfy the demand routing requirements of all source terminals. The goal of the problem is to minimize the total cost of the installed links.

The Buy-at-Bulk problem was first introduced by Salman et al. [17] in 1997. In the same year, Awerbuch and Azar [1] proposed the Selective Single-Sink Buy-at-Bulk problem, in which we only need to meet the demand routing requirements for at least k source terminals from the given set of sources, where k is an input parameter. This problem is thus called the k-SBaB problem for short in this

* Supported by NSFC 60970003, China Postdoctoral Science Foundation 20080441144 and 200902562, and the Special Foundation of Shandong Province Postdoctoral Innovation Projects with No. 200901010.

paper. Besides its practical meaning, the k-SBaB problem has its own interest since it is a generalization of both the classic k-MST problem [16,10] and the k-Steiner Tree problem [7].

Awerbuch and Azar [1] provided a randomized $O(\log^2 n)$-approximation algorithm for the k-SBaB problem, where n is the number of vertices in the input graph. Their approach to approximate k-SBaB is based on the tree-embedding method due to Bartal [2], which has wide applications in the design of approximation algorithms. The approximation ratio for the k-SBaB problem was naturally improved to $O(\log n \log\log n)$ and finally to $O(\log n)$ according to the improvements of tree-embedding by [3,4] and [8].

In this paper, we give a new deterministic approximation algorithm for the k-SBaB problem with approximation ratio $O(\sqrt{q})$, where q is the total number of source terminals. Our main contribution is a new method to approximate k-SBaB. Our approach is based on a mixed strategy of LP-rounding and the greedy method, and thus is thoroughly different from the tree-embedding approach used in [1]. To the best of our knowledge, we have not known any other non-trivial approach of approximating k-SBaB except the tree-embedding method. Although in general our approximation ratio is incomparable with the ratio $O(\log n)$ [1,8], our algorithm is obviously better than that in [1] when q is relatively small, say, $q = o(\log^2 n)$. Notice that q is always at most n. For example, when $q = \Theta(\log^{2-\epsilon} n)$ where $\epsilon > 0$ is an arbitrarily small constant, our approximation ratio is better. Note that even if $k = q = 1$ (i.e., there is only one source terminal), the problem (in the capacity-price cost model, see Section 2) is already NP-hard, since in this case it reduces to the Integer Min-Knapsack problem known to be NP-hard [9, MP10] [1, Section 3].

In fact, we can prove the integrality gap of a natural LP relaxation for k-SBaB is at least $\frac{4}{9}(q+1)$ (see Theorem 1). Our algorithm thus cannot merely rely on LP-rounding and we get around this gap by a mixed strategy of LP-rounding and greedy method. First we use the LP-rounding approach to obtain a bicriteria approximate solution to the problem. This solution is further improved to an $O(\sqrt{q})$-approximation by a simple greedy algorithm. Our LP-rounding algorithm uses the algorithm of Talwar [18] for the Buy-at-Bulk problem as a subroutine.

We briefly state some closely related works here. In general, the Buy-at-Bulk problem contains as its two subproblems the Single-Sink Buy-at-Bulk (SBaB) problem described as above, and the Multicommodity Buy-at-Bulk (MBaB) problem, with the latter having a set of source-sink pairs, instead of a set of source terminals and a sink terminal. The first constant approximation for SBaB is due to [14]. The approximation ratio was improved to 216 [18], to 76.8 [15], to 24.92 [12], and finally to 20.42 [13], which is currently the best. On the other hand, the best approximation ratio for the MBaB problem is $O(\log n)$ [1,3,8].

We should point out that all the Buy-at-Bulk problems mentioned so far, are called *uniform* in the sense that the cost function (giving the cost per unit length to route f units of flow) is the same for every edge. A more general problem is the *non-uniform* Buy-at-Bulk problem in which the cost function can vary depending on the edge. The non-uniform Buy-at-Bulk problem is beyond the scope of this paper and the readers are advised to refer to [5,6] and the references given therein for more details.

2 Preliminaries

In the following we give the formal definition of the Selective Single-Sink Buy-at-Bulk problem (i.e., the k-SBaB problem).

Instance. In the k-SBaB problem, we are given an undirected graph $G = (V, E)$ with nonnegative edge costs (i.e., lengths) $\{c_e\}$, d types of links, denoted by $L = \{0, 1, 2, \cdots, d-1\}$, a set of source terminals $D = \{s_1, s_2, \cdots, s_q\}$, a sink terminal $t \in V$, and an integer $k > 0$. Each link type $l \in L$ has a capacity $u_l > 0$ and a price per unit length $\sigma_l > 0$. The capacities and prices per unit length of all link types satisfy $u_0 \leq u_1 \leq \cdots \leq u_{d-1}$ and $\sigma_0 \leq \sigma_1 \leq \cdots \leq \sigma_{d-1}$. The cost per unit length of routing $f \geq 0$ units of flow using the link of type l is $\lceil \frac{f}{u_l} \rceil \sigma_l$. Let us call the cost model of link types given in such a way the *capacity-price* cost model. For each source $s \in D$, there is a demand $dem_s > 0$. Note that every source is also a vertex in V. An instance of k-SBaB is thus denoted by $\mathcal{I} = (G, L, D, t, k)$.

Query. We are asked to install sufficient links on edges such that at least k source terminals in D can send their demands to t in the resulting capacitated network and the total cost of the capacitated network is minimized. Let us call a solution to the k-SBaB problem a *routing scheme*, denoted by $\mathcal{S}$. For every link type l, suppose that a routing scheme $\mathcal{S}$ installs α_e^l copies of that link on edge e. To route f_e units of flow on an edge e, the total capacities $\sum_{l \in L} \alpha_e^l u_l$ of the links installed on edge e by $\mathcal{S}$ must be at least f_e. The cost on edge e paid by $\mathcal{S}$ is hence $\sum_{l \in L} \alpha_e^l \sigma_l c_e$. The total cost of routing scheme $\mathcal{S}$ is the sum of costs paid by $\mathcal{S}$ on all edges, namely, $\sum_{e \in E} (\sum_{l \in L} \alpha_e^l \sigma_l) c_e$.

In the capacity-price cost model, we may assume (Additional assumptions are required here. The readers are advised to refer to, e.g., [17], [18] and [15] for detailed argument) that the capacities and prices per unit length of all link types satisfy $\frac{\sigma_0}{u_0} \geq \frac{\sigma_1}{u_1} \geq \cdots \geq \frac{\sigma_{d-1}}{u_{d-1}}$, which reflects the economy of scale. That is, the higher the link type, the smaller the cost per unit of length per unit of capacity. While this property is not directly used in our algorithm, it is used in the algorithm of [18], which acts as a subroutine called by our algorithm.

The cost model of link types can be given in a way different from the capacity-price cost model. In the new model every link type $l \in L$ has a building cost b_l and a routing cost r_l. The building costs and the routing costs of all link types satisfy $b_0 \leq b_1 \leq \cdots \leq b_{d-1}$ and $r_0 \geq r_1 \geq \cdots \geq r_{d-1}$. The cost per unit length of routing $f > 0$ units of flow using link type l is $b_l + r_l f$. Let us call the new cost model of link types the *building-routing* cost model. Therefore, if routing scheme $\mathcal{S}$ routes $f_e^l > 0$ units of flow on an edge e using link type l, the cost paid by $\mathcal{S}$ should be $(b_l + r_l f_e^l) c_e$. The total cost of routing scheme $\mathcal{S}$ can be written as $\sum_{e \in E} (\sum_{l \in L:\, f_e^l > 0} b_l + r_l f_e^l) c_e$.

Fix any link type $l \in L$. If we set $b_l = \sigma_l$ and $r_l = \frac{\sigma_l}{u_l}$, then it is easy to see that the routing cost per unit length function $h(f) = b_l + r_l f$ in the building-routing cost model is always within a factor 2 of the routing cost per unit length function $g(f) = \lceil \frac{f}{u_l} \rceil \sigma_l$ in the capacity-price cost model, i.e., we always have $g(f) \leq h(f) \leq 2g(f)$ for all $f > 0$ [11]. This means that the above reduction preserves costs up to a factor of 2. Therefore, we assume from now on that we are dealing with the k-SBaB problem in the building-routing cost model of link types.

The following Lemma 1 states a well-known property about the routing scheme to the Buy-at-Bulk problem.

Lemma 1. *Suppose that a routing scheme $\mathcal{S}$ uses more than 1 link types on some edge e. Then there exists a routing scheme $\mathcal{S}'$ which routes the same amount of flow for every source terminal as what does $\mathcal{S}$, such that* $\text{cost}(\mathcal{S}')$ *is at most* $\text{cost}(\mathcal{S})$ *and $\mathcal{S}'$ uses only 1 link type on edge e.*

Proof. Suppose the routing scheme $\mathcal{S}$ uses h link types $\{l_1, l_2, \cdots, l_h\}$ on edge e with flow values $f_1, f_2, \cdots, f_h$ on each link type, respectively. The cost of $\mathcal{S}$ paid on e is $\sum_{i=1}^{h} b_{l_i} + r_{l_i} f_i$. Define function $g(f) = \min_i \{b_{l_i} + r_{l_i} f\}$. Then $g(f) \leq b_{l_i} + r_{l_i} f$ for all f and i. Moreover, since each function $b_{l_i} + r_{l_i} f$ is linear, $g(f)$ is concave. So we have $g(\sum_i f_i) \leq \sum_{i=1}^{h} b_{l_i} + r_{l_i} f_i$ and the lemma follows. □

3 LP Formulation

We get our LP relaxation for the k-SBaB problem by modifying the one used in [11,18] for the SBaB problem. The LP relaxation for k-SBaB is given as (LP$_k$).

As what was done in [11], we replace each undirected edge in G by a pair of anti-parallel directed edges, each having the same length as the original one. The LP relaxation (LP$_k$) is based on the so modified instance. For a vertex $v \in V$, we use $\text{in}(v)$ to denote the set of directed edges pointing to v, i.e., $\text{in}(v) = \{(u,v) \mid (u,v) \in E\}$. Similarly, $\text{out}(v) = \{(v,w) \mid (v,w) \in E\}$ is the set of directed edges leaving v.

In constraint (6), if we let $x_{e;s}^l, y_s, z_e^l$ be 0-1 integers, then we get the LP formulation (namely, the integer linear program) for the k-SBaB problem, denoted by (IP$_k$). Let us consider (IP$_k$) now. $y_s = 1$ means that we may leave source s as an outlier. Constraint (2) states that the number of outliers cannot exceed $q - k$, that is, we must connect at least k source terminals to sink t. $x_{e;s}^l = 1$ means we should deploy link type l on edge e for source s. Let $x_{e;s} = \sum_{l=0}^{d-1} x_{e;s}^l$. Constraint (1) and constraint (3) mean that in the optimal solution, for every source terminal s, the edges with $x_{e;s} \geq 1$ must constitute exactly an s-t path. Constraint (4) means that along any s-t path, the installed link types are non-decreasing, which is a property possessed by the optimal solution to the k-SBaB problem under the building-routing cost model [11, Theorem 1]. $z_e^l = 1$ means that we should install link type l on edge e (and thus we must pay the building cost $b_l c_e$). Then constraint (5) indicates when we need to pay the building cost. The object function of (IP$_k$) is to minimize the sum of the total building cost and the total routing cost.

$$\min \sum_{e\in E}\sum_{l=0}^{d-1} b_l z_e^l c_e + \sum_{s\in D}\sum_{e\in E}\sum_{l=0}^{d-1} dem_s x_{e;s}^l r_l c_e \qquad (\text{LP}_k)$$

$$\text{s.t.} \quad \sum_{e\in \text{out}(s)}\sum_{l=0}^{d-1} x_{e;s}^l + y_s \geq 1 \qquad \forall s \in D \qquad (1)$$

$$\sum_{s\in D} y_s \leq q-k \qquad (2)$$

$$\sum_{e\in \text{in}(v)}\sum_{l=0}^{d-1} x_{e;s}^l = \sum_{e\in \text{out}(v)}\sum_{l=0}^{d-1} x_{e;s}^l \qquad \forall s \in D, \forall v \in V\setminus\{s,t\} \qquad (3)$$

$$\sum_{e\in \text{in}(v)}\sum_{l=d'}^{d-1} x_{e;s}^l \leq \sum_{e\in \text{out}(v)}\sum_{l=d'}^{d-1} x_{e;s}^l \qquad \forall s \in D, \forall v \in V\setminus\{s,t\}, \; \forall 0 \leq d' \leq d-1 \qquad (4)$$

$$x_{e;s}^l \leq z_e^l \qquad \forall l \in L, \forall e \in E, \forall s \in D \qquad (5)$$

$$x_{e;s}^l, y_s, z_e^l \geq 0 \qquad \forall l \in L, \forall e \in E, \forall s \in D \qquad (6)$$

For the LP relaxation (LP$_k$), it is convenient to think of $x_{e;s}^l$ as the flow value deployed by source terminal s on edge e using link type l (although the real flow value should be $dem_s x_{e;s}^l$). Then constraint (3) is essentially the flow conservation constraint.

For LP relaxation (LP$_k$) we have Theorem 1.

Theorem 1. *The integrality gap of the natural linear program (LP$_k$) for k-SBaB is at least $\frac{4}{9}(q+1)$.*

Proof. Consider the instance $\mathcal{I}$ in Figure 1. In this instance, we have two link types, namely, $L = \{0,1\}$, with

$$\begin{cases} b_0 = 1, r_0 = 1 & \text{link type } 0 \\ b_1 = 2, r_1 = \frac{1}{2q} & \text{link type } 1 \end{cases}$$

For every source terminal $s \in D$, $dem_s = 1$. In addition, k is set to be 1, that is, we need only connect one source terminal to t. Let E_t be the set of all edges at the top, and E_b be the set of all edges at the bottom.

Since all the source terminals are essentially identical, the optimal solution to this instance can connect any source terminal, say s_1, to t. The best way to route the unit demand of s_1 to t is to deploy link type 0 on the path from s_1 to t. Thus the cost $\text{OPT}_k(\mathcal{I})$ of the optimal solution to instance $\mathcal{I}$ is $(b_0 + r_0 \cdot dem_1)(c_{e_1} + c_{e_2}) = (1 + 1\cdot 1)(1 + \frac{1}{q}) = 2 + \frac{2}{q}$.

Then consider an (optimal) fractional solution to (LP$_k$) on instance $\mathcal{I}$. For all $s \in D$, we set $y_s = 1 - \frac{1}{q}$. To satisfy constraint (1), each source terminal $s \in D$ need only to send $\frac{1}{q}$ unit of flow to t; this can be done by sending $\frac{1}{pq}$ unit of flow on each of the p paths from s to t consisting of only 2 edges.

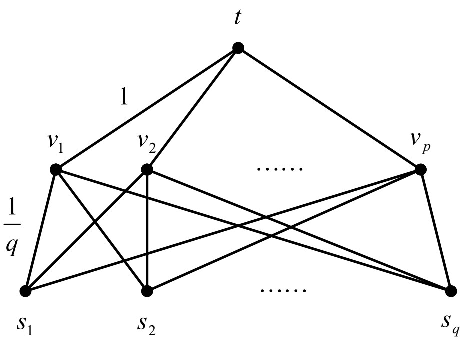


Fig. 1. An instance of k-SBaB showing that the integrality gap of (LP$_k$) is at least $\frac{4}{9}(q+1)$

For the bottom edges, we deploy link type 0 to carry the $\frac{1}{q}$ unit of flow for every source terminal, that is, for each edge $e \in E_b$ and each source $s \in D$, we set $x^{(0)}_{e;s} = \frac{1}{pq}$ and $z^{(0)}_e = \frac{1}{pq}$. The building cost on the bottom edges is

$$\sum_{e \in E_b} b_0 \cdot z^{(0)}_e \cdot c_e = (pq)(1 \cdot \frac{1}{pq} \cdot \frac{1}{q}) = \frac{1}{q}.$$

The routing cost on the bottom edges is

$$\sum_{e \in E_b} \sum_{s \in D} dem_s \cdot x^{(0)}_{e;s} \cdot r_0 \cdot c_e = (pq)(1 \cdot \frac{1}{pq} \cdot 1 \cdot \frac{1}{q}) = \frac{1}{q}.$$

Thus the total cost paid on the bottom edges is $\frac{2}{q}$. This is the best way to send the $\frac{1}{q}$ unit of flow for every source terminal to the top edges. (One can verify that if we deploy link type 1 on the bottom edges to carry the flow for all sources, the total cost would be $\frac{2}{q} + \frac{1}{2q^2}$.)

For the top edges, we deploy link type 1 to carry the $\frac{1}{q}$ unit of flow for every source terminal. For each edge $e \in E_t$ and each source $s \in D$, we set $x^{(1)}_{e;s} = \frac{1}{pq}$ and $z^{(1)}_e = \frac{1}{pq}$. The building cost on the top edges is

$$\sum_{e \in E_t} b_1 \cdot z^{(1)}_e \cdot c_e = p(2 \cdot \frac{1}{pq} \cdot 1) = \frac{2}{q}.$$

The routing cost on the top edges is

$$\sum_{e \in E_t} \sum_{s \in D} dem_s \cdot x^{(1)}_{e;s} \cdot r_1 \cdot c_e = (p \cdot q)(\frac{1}{pq} \cdot \frac{1}{2q} \cdot 1) = \frac{1}{2q}.$$

Thus the total cost paid on the top edges is $\frac{1}{2q} + \frac{2}{q}$. Furthermore, this is the best way to send the $\frac{1}{q}$ unit of flow for every source terminal from the middle vertices to t. (One can verify that if we deploy link type 0 on the top edges to carry the flow for all sources, the total cost would be $1 + \frac{1}{q}$.)

Therefore, the cost $\mathrm{OPT}_f(\mathrm{LP}_k(\mathcal{I}))$ of the optimal fractional solution to (LP_k) on instance $\mathcal{I}$ is $\frac{2}{q} + \frac{1}{2q} + \frac{2}{q} = \frac{4}{q} + \frac{1}{2q}$. This shows that the integrality gap of (LP_k) is at least $\frac{\mathrm{OPT}_k(\mathcal{I})}{\mathrm{OPT}_f(\mathrm{LP}_k(\mathcal{I}))} = \frac{2+2/q}{4/q+1/2q} = \frac{4}{9}(q+1)$. □

In our algorithm given in Section 4, we shall use the LP relaxation for the Buy-at-Bulk problem, as shown in the following linear program (LP_s).

$$\min \sum_{e\in E}\sum_{l=0}^{d-1} b_l z_e^l c_e + \sum_{s\in D}\sum_{e\in E}\sum_{l=0}^{d-1} dem_s x_{e;s}^l r_l c_e \qquad (\mathrm{LP}_s)$$

$$\text{s.t.} \sum_{e\in \mathrm{out}(s)}\sum_{l=0}^{d-1} x_{e;s}^l \ge 1 \qquad \forall s \in D \qquad (7)$$

$$\sum_{e\in \mathrm{in}(v)}\sum_{l=0}^{d-1} x_{e;s}^l = \sum_{e\in \mathrm{out}(v)}\sum_{l=0}^{d-1} x_{e;s}^l \qquad \forall s \in D, \forall v \in V\setminus\{s,t\} \qquad (8)$$

$$\sum_{e\in \mathrm{in}(v)}\sum_{l=d'}^{d-1} x_{e;s}^l \le \sum_{e\in \mathrm{out}(v)}\sum_{l=d'}^{d-1} x_{e;s}^l \qquad \forall s \in D, \forall v \in V\setminus\{s,t\}, \quad \forall 0 \le d' \le d-1 \qquad (9)$$

$$x_{e;s}^l \le z_e^l \qquad \forall l\in L, \forall e \in E, \forall s\in D \qquad (10)$$

$$x_{e;s}^l, z_e^l \ge 0 \qquad \forall l\in L, \forall e \in E, \forall s\in D$$

4 Algorithm

4.1 A Simple Greedy Algorithm

Our first algorithm for k-SBaB is a simple greedy algorithm, shown as algorithm $\mathcal{A}$.

```
   Algorithm $\mathcal{A}$ for $k$-SBaB
   Input: an instance $(G, L, D, t, k)$ of k-SBaB.
   Output: a routing scheme connecting to sink $t$ at least $k$ source terminals
     in $D$.
1  for each source terminal $s \in D$ do
2     Compute a shortest path $p$ from $s$ to $t$.
3     Choose link type $l \in L$ such that $b_l + r_l dem_s$ is minimized.
4     Deploy link type $l$ on path $p$ with flow value $dem_s$. This is our routing
        scheme $\mathcal{S}_s$ for source terminal $s$.
5  endfor
6  return the union of the first $k$ cheapest routing schemes as the final solution
     $\mathcal{S}$.
```

A routing scheme $\mathcal{S}$ can be viewed as a set of tuples $\{(e, l, f_e^l), \cdots\}$, where (e, l, f_e^l) means that the scheme $\mathcal{S}$ deploys link type l on edge e with flow value f_e^l. In step 6 of algorithm $\mathcal{A}$, the union of the routing schemes is done in the natural way similar to set union. Notice that by Lemma 1, we can assume that the final routing scheme found by algorithm $\mathcal{A}$ uses only 1 link type on each used edge.

For algorithm $\mathcal{A}$ we have Lemma 2.

Lemma 2. *Algorithm $\mathcal{A}$ is a k-approximation algorithm for the k-SBaB problem.*

Proof. Obviously algorithm $\mathcal{A}$ runs in polynomial time.

Notice that steps 2, 3 and 4 of algorithm $\mathcal{A}$ actually computes an optimal solution $\mathcal{S}_s$ to instance $(G, L, \{s\}, t)$ of the Buy-at-Bulk problem, in which there is only one source terminal and the link types are given under the building-routing cost model. Let $\mathcal{S}^*$ be an optimal solution to the k-SBaB problem and OPT_k be its solution value. For each source terminal s routed by $\mathcal{S}^*$, we have $\mathrm{cost}(\mathcal{S}_s) \le \mathrm{OPT}_k$. Since the routing scheme $\mathcal{S}$ is the union of the first k cheapest routing schemes in $\{\mathcal{S}_s \mid s \in D\}$, we have $\mathrm{cost}(\mathcal{S}) \le k \cdot \mathrm{OPT}_k$. This concludes the lemma. □

4.2 A Bicriteria LP-Rounding Algorithm

Our second algorithm, shown as algorithm $\mathcal{B}$, is a LP-rounding algorithm that outputs a bicriteria solution to the k-SBaB problem. The input parameter $\epsilon > 0$ of algorithm $\mathcal{B}$ is a small number and will be set by algorithm $\mathcal{C}$ in Subsection 4.3.

```
Algorithm $\mathcal{B}$ for $k$-SBaB
Input: an instance $(G, L, D, t, k)$ of $k$-SBaB, and a parameter $\epsilon > 0$.
Output: a routing scheme that connects to $t$ at least $(1-\epsilon)k$ source terminals
  in $D$.
1 Find an optimal solution $(x, y, z)$ to $(\mathrm{LP}_k)$.
2 $\alpha \leftarrow \frac{q-k}{q-(1-\epsilon)k}$.
3 for each source terminal $s \in D$ do
4     if $y_s \ge \alpha$ then $\bar{y}_s \leftarrow 1$ else $\bar{y}_s \leftarrow 0$.
5 endfor
6 $C \leftarrow \{s \colon \bar{y}_s = 0\}$.
7 Find an integral feasible solution $(\bar{x}, \bar{z})$ to $(\mathrm{LP}_s)$ on $C$ by [18].
8 return $(\bar{x}, \bar{y}, \bar{z})$.
```

In step 7 of algorithm $\mathcal{B}$, (LP_s) on C means the linear program (LP_s) with source terminal set D replaced by C.

Lemma 3. *Given any small number $\epsilon \in (0, 1)$ (not necessarily fixed) such that the rounding threshold α can be computed in polynomial time, algorithm $\mathcal{B}$ outputs in polynomial time a solution to the instance of k-SBaB which connects to t at least $(1-\epsilon)k$ source terminals in D at the cost of at most $\frac{c_0}{\epsilon} \cdot \frac{q-k+\epsilon k}{k} \cdot \mathrm{OPT}_k$, where c_0 is some constant, and OPT_k is the optimum of the instance.*

Proof. If for some source terminal s we have $y_s > 1$ in the optimal solution (x, y, z), we can decrease y_s to 1, not violating constraints (1) and (2). Therefore, without loss of generality, we can assume that $y_s \leq 1$ for all $s \in D$. Similarly, due to the minimization of the objection function of (LP$_k$), we conclude that $x^l_{e;s} \leq 1$ for all $s \in D$, $e \in E$ and $l \in L$, and that constraint (1) holds with equality for the optimal solution (x, y, z).

Since $0 < \epsilon < 1$, the rounding threshold α is in $(0, 1)$. Algorithm $\mathcal{B}$ connects all source terminals in C by leaving the source terminals such that $y_s \geq \alpha$ alone.

Let s be any but fixed terminal in C. Define $\beta_s = \frac{1}{1-y_s}$ and $\hat{x}^l_{e;s} = \beta_s x^l_{e;s}$ for all $e \in E$ and $l \in L$. Since $\sum_{e\in \text{out}(s)} \sum_{l=0}^{d-1} x^l_{e;s} = 1 - y_s$, we have $\sum_{e\in \text{out}(s)} \sum_{l=0}^{d-1} \hat{x}^l_{e;s} = 1$ and hence constraint (7) of (LP$_s$) holds. Since all $\hat{x}^l_{e;s}$'s (note that s is fixed) are amplified by the same factor β_s, we know that $\hat{x}^l_{e;s}$'s satisfy constraints (8) and (9). Moreover, every $\hat{x}^l_{e;s}$ is at most 1, since the total amount of $x^l_{e;s}$'s emerging from s is $1 - y_s$, and the amount of $x^l_{e;s}$'s flowing into a vertex $v \neq s, t$ and the amount of that flowing out of v are equal by the flow conservation constraint (3).

We define β_s and $\hat{x}^l_{e;s}$ for every source $s \in C$ as above. Finally, Let $\beta = \max_s\{\beta_s\}$ and define $\hat{z}^l_e = \beta z^l_e$. Then constraint (10) is satisfied by $(\hat{x}, \hat{z})$. This shows that $(\hat{x}, \hat{z})$ is a feasible solution to (LP$_s$) on C.

By [18], an integral c_0-approximate solution $(\bar{x}, \bar{z})$ to (LP$_s$) on C can be found in polynomial time, where c_0 is a constant given in [18] ($c_0 \leq 108$). By definition, we have $\beta_s = \frac{1}{1-y_s} \leq \frac{1}{1-\alpha}$ for every $s \in C$ and $\beta \leq \frac{1}{1-\alpha}$. So we get an integral solution $(\bar{x}, \bar{y}, \bar{z})$ to (LP$_k$) satisfying

$$\begin{aligned}
&\sum_{e\in E}\sum_{l\in L} b_l \bar{z}^l_e c_e + \sum_{s\in C}\sum_{e\in E}\sum_{l\in L} dem_s \bar{x}^l_{e;s} r_l c_e \\
&\leq c_0 \text{OPT}_f(\text{LP}_s) \\
&\leq c_0 \Big(\sum_{e\in E}\sum_{l\in L} b_l \hat{z}^l_e c_e + \sum_{s\in C}\sum_{e\in E}\sum_{l\in L} dem_s \hat{x}^l_{e;s} r_l c_e\Big) \\
&\leq \frac{c_0}{1-\alpha} \Big(\sum_{e\in E}\sum_{l\in L} b_l z^l_e c_e + \sum_{s\in C}\sum_{e\in E}\sum_{l\in L} dem_s x^l_{e;s} r_l c_e\Big) \\
&\leq \frac{c_0}{1-\alpha} \text{OPT}_k.
\end{aligned}$$

Denote by Y the number of source terminals which are not connected to t by the solution $(\bar{x}, \bar{y}, \bar{z})$, i.e., $Y = \sum_{s\in D} \bar{y}_s$. We claim that $Y \leq q - (1 - \epsilon)k$. Otherwise it would be the case that $\sum_{s\in D} y_s > (q - (1 - \epsilon)k) \cdot \alpha = q - k$ and hence the constraint (2) of (LP$_k$) is violated. So the solution $(\bar{x}, \bar{z})$ connects to t at least $q - Y \geq (1 - \epsilon)k$ source terminals in D.

Given that the rounding threshold α can be computed in polynomial time, we know that algorithm $\mathcal{B}$ runs in polynomial time. This concludes the lemma. □

4.3 An $O(\sqrt{q})$-Approximation Algorithm

We have already known that for any $1 > \epsilon > 0$, algorithm $\mathcal{B}$ will give a bicriteria solution which connects to t at least $(1-\epsilon)k$ source terminals. If it actually connects at least k source terminal, then we are done. Otherwise we have to connect additional ϵk source terminals to t to obtain a feasible solution to the k-SBaB problem. But we just have the greedy algorithm $\mathcal{A}$ to complete this task, since the task is in fact a new k-SBaB instance. This idea gives us the final algorithm $\mathcal{C}$.

```
Algorithm $\mathcal{C}$ for $k$-SBaB
Input: an instance $\mathcal{I} = (G, L, D, t, k)$ of k-SBaB.
Output: a routing scheme that connects to t at least k source terminals in
  D.
 1 if $k \le \sqrt{q}$ then
 2   call algorithm $\mathcal{A}$, obtaining a routing scheme $\mathcal{S}$.
 3   return $\mathcal{S}$ as the final solution to $\mathcal{I}$ and stop.
 4 endif
 5 call algorithm $\mathcal{B}$ with $\epsilon = \frac{\sqrt{q}}{k}$, obtaining a routing scheme $\mathcal{S}'$. Let $D'$ be
     the set of source terminals connected by $\mathcal{S}'$.
 6 $k' \leftarrow |D'|$.
 7 if $k' \ge k$ then
 8   return $\mathcal{S}'$ as the final solution to $\mathcal{I}$ and stop.
 9 endif
10 $D_0 \leftarrow D - D'$, $k_0 \leftarrow k - k'$. This defines a new instance $\mathcal{J} = (G, L, D_0, t, k_0)$
     of $k$-SBaB.
11 call algorithm $\mathcal{A}$ on instance $\mathcal{J}$, obtaining a solution $\mathcal{S}_0$.
12 Let routing scheme $\mathcal{S}''$ be the union of $\mathcal{S}'$ and $\mathcal{S}_0$. return $\mathcal{S}''$ as the final
     solution to $\mathcal{I}$.
```

Theorem 2. *Algorithm $\mathcal{C}$ is an $O(\sqrt{q})$-approximation algorithm for the k-SBaB problem.*

Proof. If $k \le \sqrt{q}$, algorithm $\mathcal{C}$ will return routing scheme $\mathcal{S}$ as the final solution, which is found by the greedy algorithm $\mathcal{A}$. By Lemma 2, $\mathcal{S}$ is a k-approximate solution to instance $\mathcal{I}$. The theorem is proved under this case. So in the following we assume that $k > \sqrt{q}$.

Since $k > \sqrt{q}$ and $\epsilon = \frac{\sqrt{q}}{k}$, we know that $1 > \epsilon > 0$. Recall that k' is the number of source terminals connected by $\mathcal{S}'$. Denote by $\mathrm{OPT}_k(\mathcal{I})$ the optimum of instance $\mathcal{I}$. By Lemma 3, for the routing scheme $\mathcal{S}'$ and the number k' we have

$$\mathrm{cost}(\mathcal{S}') \le \frac{c_0(q-(1-\epsilon)k)}{\epsilon k} \cdot \mathrm{OPT}_k(\mathcal{I}) \le c_0\sqrt{q} \cdot \mathrm{OPT}_k(\mathcal{I})$$

and

$$k' \ge (1-\epsilon)k.$$

If it happens that $k' \ge k$, algorithm $\mathcal{C}$ will return $\mathcal{S}'$ and the theorem follows.

Then let us consider the case $k' < k$. In this case, algorithm $\mathcal{C}$ calls the greedy approximation algorithm $\mathcal{A}$ to obtain a routing scheme $\mathcal{S}_0$ to the new instance $\mathcal{J} = (G, L, D_0, t, k_0)$. Since $\mathcal{S}_0$ connects to t at least $k_0 = k - k'$ source terminals in $D_0 = D - D'$, it is obvious that the routing scheme $\mathcal{S}''$, which is the union of $\mathcal{S}'$ and $\mathcal{S}_0$, is a feasible solution to instance $\mathcal{I}$.

Suppose that $D^* \subseteq D$ is the set of source terminals connected by an optimal solution $\mathcal{S}^*$ to instance $\mathcal{I}$. Since $|D'| = k' < k$ and $|D^*| \geq k$, there are at least $k_0 = k - k'$ source terminals in $D^* - D' \subseteq D_0$. Since $\mathcal{S}^*$ obviously connects all source terminals in $D^* - D'$ to t, $\mathcal{S}^*$ is a feasible solution to instance $\mathcal{J}$. It turns out that $\mathrm{OPT}(\mathcal{J}) \leq \mathrm{cost}(\mathcal{S}^*) = \mathrm{OPT}_k(\mathcal{I})$, where $\mathrm{OPT}(\mathcal{J})$ is the optimum of instance $\mathcal{J}$. By Lemma 2, $\mathcal{S}_0$ is a k_0-approximate solution to instance $\mathcal{J}$. So we get that $\mathrm{cost}(\mathcal{S}_0) \leq k_0 \cdot \mathrm{OPT}(\mathcal{J}) \leq k_0 \cdot \mathrm{OPT}_k(\mathcal{I})$. Thus for the routing scheme $\mathcal{S}''$ we have

$$\begin{aligned}\mathrm{cost}(\mathcal{S}'') &\leq \mathrm{cost}(\mathcal{S}') + \mathrm{cost}(\mathcal{S}_0)\\ &\leq (c_0\sqrt{q} + k_0) \cdot \mathrm{OPT}_k(\mathcal{I})\\ &\leq (c_0\sqrt{q} + \epsilon k) \cdot \mathrm{OPT}_k(\mathcal{I})\\ &= (c_0 + 1)\sqrt{q} \cdot \mathrm{OPT}_k(\mathcal{I}).\end{aligned}$$

Algorithm $\mathcal{A}$ runs in polynomial time. When ϵ is set to $\frac{\sqrt{q}}{k}$, the rounding threshold α used in algorithm $\mathcal{B}$ obviously can be computed in polynomial time. So algorithm $\mathcal{B}$ runs in polynomial time. It follows that algorithm $\mathcal{C}$ is in polynomial time. The theorem follows. $\square$

References

1. Awerbuch, B., Azar, Y.: Buy-at-bulk network design. In: Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 542–547 (1997)
2. Bartal, Y.: Probabilistic approximations of metric spaces and its algorithmic applications. In: Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS), pp. 184–193 (1996)
3. Bartal, Y.: On approximating arbitrary metrics by tree metrics. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), pp. 161–168 (1998)
4. Charikar, M., Chekuri, C., Goel, A., Guha, S., Plotkin, S.A.: Approximating a finite metric by a small number of tree metrics. In: Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS), pp. 379–388 (1998)
5. Charikar, M., Karagiozova, A.: On non-uniform multicommodity buy-at-bulk network design. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 176–182 (2005)
6. Chekuri, C., Hajiaghayi, M., Kortsarz, G., Salavatipour, M.: Approximation algorithms for nonuniform buy-at-bulk network design. SIAM Journal on Computing 39(5), 1772–1798 (2010)
7. Chudak, F.A., Roughgarden, T., Williamson, D.P.: Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. Math. Program. 100(2), 411–421 (2004)

8. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. Journal of Computer and System Sciences 69(3), 485–497 (2004)
9. Garey, M., Johnson, D.: Computers and Intractability. W.H.Freeman and Company, New York (1979)
10. Garg, N.: Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), pp. 396–402 (2005)
11. Garg, N., Khandekar, R., Konjevod, G., Ravi, R., Salman, F.S., Sinha, A.: On the integrality gap of a natural formulation of the single-sink buy-at-bulk network design problem. In: Aardal, K., Gerards, B. (eds.) IPCO 2001. LNCS, vol. 2081, pp. 170–184. Springer, Heidelberg (2001)
12. Grandoni, F., Italiano, G.: Improved approximation for single-sink buy-at-bulk. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 111–120. Springer, Heidelberg (2006)
13. Grandoni, F., Rothvoß, T.: Network design via core detouring for problems without a core. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 490–502. Springer, Heidelberg (2010)
14. Guha, S., Meyerson, A., Munagala, K.: A constant factor approximation for the single sink edge installation problems. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), pp. 383–388 (2001)
15. Gupta, A., Kumar, A., Pál, M., Roughgarden, T.: Approximation via cost sharing: simpler and better approximation algorithms for network design. Journal of the ACM 54(3), Article 11 (2007)
16. Ravi, R., Sundaram, R., Marathe, M.V., Rosenkrantz, D.J., Ravi, S.S.: Spanning trees short or small. In: Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 546–555 (1994)
17. Salman, F., Cheriyan, J., Ravi, R., Subramanian, S.: Buy-at-bulk network design: approximating the single-sink edge installation problem. In: Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 619–628 (1997)
18. Talwar, K.: The single-sink buy-at-bulk LP has constant integrality gap. In: Cook, W., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 475–480. Springer, Heidelberg (2002)

Greedy Algorithm for Least Privilege in RBAC Model*

Jinling Liu, Hejiao Huang, and Hongwei Du

Harbin Institute of Technology Shenzhen Graduate School, China
jdouble12007@163.com, hejiao0@yahoo.com.cn, hongwei.du@ieee.org

Abstract. Least privilege means only the necessary privileges are needed to complete a task for users. This is one of the most important principles in RBAC model. Currently, how to assign roles to users to achieve this principle is still not solved. In this paper, the least privilege problem is proved to be NP-hard, and an approximation algorithm is given. The simulation result shows that with the algorithm, each user can acquire its privilege to perform its job with the least privilege principle.

Keywords: Least privilege, NP-hard, greedy algorithm, performance ratio.

1 Introduction

Role-Based Access Control (RBAC) is an approach for improving computer system security. Based on RBAC technology standard, a set of specific privileges can perform a specific operation, and the system user should be assigned to particular roles which have certain privileges in order to perform system operations [4]. In other words, the user can not get any privileges directly, instead, can acquire them through roles. In RBAC, a user can perform certain operations only if the user has all the necessary privileges to perform certain operations through roles. For example, in Fig. 1, suppose user u_2 should have the privilege p_1, p_2 and p_4 (Fig. 1(a)) in order to complete a particular operation. Based on the standard, u_2 cannot get these privileges directly, instead, it can be assigned to role r_1 which has these privileges (Fig. 1).

Based on RBAC technology standard, each role is given privileges in advance. Users can obtain their privileges through roles in RBAC, but it is unsafe to give users more privileges than they really need. For example, in Fig. 1(b), user u_2 is assigned to role r_1, so u_2 has all the privileges that r_1 has, including the redundant privilege p_5. As a result, u_2 has a chance to execute some operations related to p_5 which leads to some system security problems, e.g., leaking important information. In order to decrease or avoid such kind of security risk, least privilege principle should be considered for user-role assignment.

Least privilege principle means a user should not have too much more privileges than necessity to operate in his task. In other words, a user should complete his task with redundant privileges as little as possible. For example, in Fig. 1, suppose the user

* This work was financially supported by National Natural Science Foundation of China under Grant No.11071271, the Fundamental Research Funds for the Central Universities under Grant No. HIT. NSRIF. 2009 and Shenzhen Special Funds under grant No. CXB201005250026A.

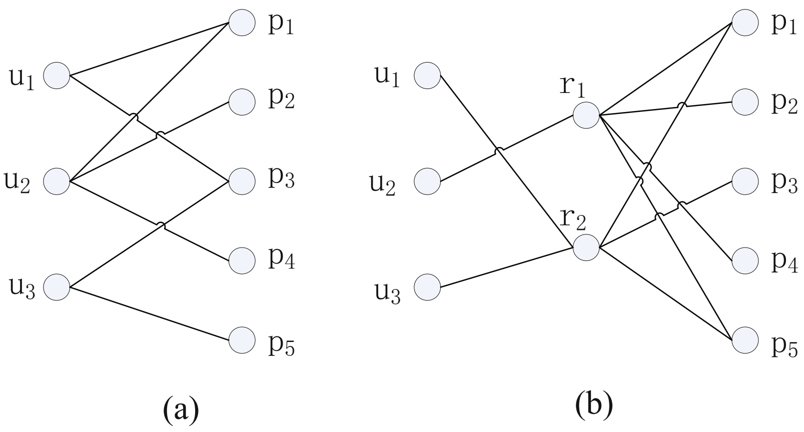


Fig. 1. Users get privileges through roles

u_1 should have privilege p_1 and p_3 (Fig. 1(a)) in order to do a special task. Based on this principle, assigning role r_2 to u_1 is better than assigning both r_1 and r_2 to u_1 for reducing redundant privileges.

The least privilege principle is one of the most important security policies for systems in RBAC, but it is still a problem to be solved. Previous research has done much related work. For example,core RBAC and Hierarchical RBAC has been introduced [4], and in each RBAC model, there are sessions between roles and users. If a user wants to obtain privileges, the session between the user and the assigned roles who contain all of those privileges must be activated. Though the session part can control which roles can be used by the user, it cannot make sure the user-role assignment follows the least privilege principle. Later, Chen and Crampton uses sessions to implement the principle of least privilege in RBAC96 [3], but it is used to solve inter-domain role mapping and it is hard to return an optimal solution except in special cases. Barka and Sandhu [1,2] have proposed role-to-role delegation mechanism, and considered delegation and revocations with multi-step. Then Wainer and Kumar [10] have put forward user-to-user delegation model, and set delegation conditions to control multi-step delegations and revocations. A new delegation model to implement the principle of least privilege has been proposed in [5]. In this model, dynamic delegation mechanism and credit mechanism are used. Although these delegation-based approaches considered least privilege problem, which still cannot ensure user-role assignment follows the principle of least privilege. In addition, the delegation constrains are complex to manage.

In order to protect sensitive privileges, and then control redundant privileges, Mu Xiaojun [12] has proposed a boundary-based access control model, but the mechanism is hard to carry out in RBAC model owing to its complexity. There is also much work about role mining in previous research, e.g., Vaidya et al. [9] and Schlegelmilch et al. [8] consider role mining using subset enumeration and cluster, respectively. However, these methods just involve how to solve assignment between roles and privileges, not assignment between roles and users. Hence, the principle of least privilege is not really implemented. Later, Lai gives a quantitative analysis and measurement for the principle of least privilege [6]. However, these previous works neither solve the problem of least privilege nor measure the hardness of it.

This paper firstly proves that least privilege problem in user-role assignment is NP-hard and an approximation algorithm is presented. Through the approximation algorithm, roles can be assigned to different users. At the same time, some redundant roles can be removed.

The remaining part of this paper is arranged as follows: Section 2 introduces the least privilege problem of user-role assignment and it is proved to be NP-hard. In Section 3, the approximation algorithm is given. In addition, the performance analysis is shown. The simulation result helps to analyze whether users can get their own roles with the principle of least privilege in a short time. Section 4 concludes this paper.

2 Least Privilege User-Role Assignment Problem

The principle of least privileges requires that the user is assigned with no more privileges than necessity to complete a task or job. This principle makes sure that a system is safe in some degree. So assigning roles to the user needs to follow this principle. According to user's task or job, the privileges that assigned to user are divided into two parts: target privilege and non-target privilege [9]. Target privilege is necessary to complete a task or job, and non-target privilege is not. However, the user just needs target privileges, so roles assigned to the user must cover all the target privileges, with non-target privileges as little as possible.

How to assign roles to the user with least privilege principle is the major problem. In fact, it is NP-hard and will be proved later.

2.1 Problem Formulation

Let $R = \{r_1, r_2, \cdots, r_m\}$ denote the set of roles, $P = \{p_1, p_2, \cdots, p_n\}$ denote the set of privileges, and T denote the set of target privileges for a user. Construct a bipartite graph $G = (R \cup P, E)$: R and P are node sets representing the roles and the privileges respectively as defined above, if $r_i \in R$ contains privilege $p_j \in P$, there is an edge $(r_i, p_j) \in E$ (Fig. 2).

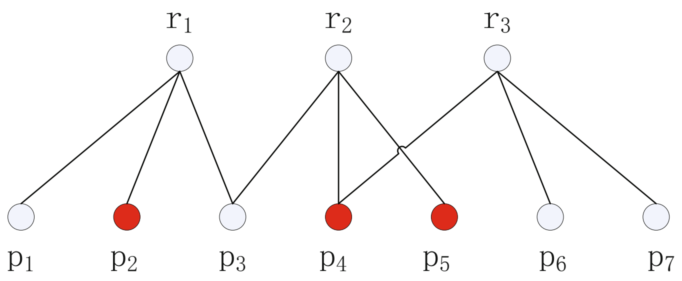


Fig. 2. Relations between roles and privileges

In Fig. 2, $R = \{r_1, r_2, r_3\}$, $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$, there are 9 edges between R and P. Let B denotes a subset of R. $N(B)$ is the neighbor nodes of B. For example, if $B = \{r_1, r_2\}$, then $N(B) = \{p_1, p_2, p_3, p_4, p_5\}$ in Fig. 2.

Now the concept of weight [6] should be introduced. Each privilege can be given a weight value, which implies the important index for the system security. The larger the weight value is, the more important for the system security. For example,"Changing the students' information" is more important than "reading the students' information". In other words, the weight value is larger, the privilege is more dangerous to leak the system information. Hence, "Changing the students' information" has larger weight than "reading the students' information".

Each privilege has its own weight value, and let ω denote the weight value. Suppose privilege p_i has weight ω_i.

Let function c denote the sum of non-target privileges' weights, that is $r \in R$, $c(r) = \sum_{p_i \in (N(r)-T)} \omega_i$. For example, in Fig. 2, $T = \{p_2, p_4, p_5\}$, $N(r_1) = \{p_1, p_2, p_3\}$ but the only target privilege is p_2 , so $c(r_1) = \omega_1 + \omega_3$.

Similarly, define the function for the subset B as follows: $c(B) = \sum_{p_i \in (N(B)-T)} \omega_i$. The function $c(B)$ is called cost function.

Now define another important function $f(B) = |N(B) \cap T|$ which means the number of target privileges that $N(B)$ has.

The least privilege problem is to find: $\min c(B) : f(B) = f(R), B \subseteq R$ on 2^R, which means non-target privileges obtained are as little as possible and all the target privileges are covered.

2.2 Minimum Submodular Cover with Submodular Cost

Suppose that f is a polymatroid functions on 2^E. A polymatroid function f has three factors [11]:

1. f is increasing;
2. f is submodular, that is for any two subsets B_1 and B_2 of R,
   $f(B_1) + f(B_2) \geq f(B_1 \cup B_2) + f(B_1 \cap B_2)$;
3. $f(\emptyset) = 0$.

Then, a set $X \in E$ is said to be a submodular cover of (E, f) if $f(X) = f(E)$. Suppose that both f and c are polymatroid functions on 2^E. The minimization problem $\min c(X) : f(X) = f(E), X \in E$ is known as a Minimum Submodular Cover with Submodular Cost [11].

In the following, how to achieve least privilege is proved to be NP-hard. It will be completed with the help of Minimum Submodular Cover with Submodular Cost problem, which is NP-hard [11].

Lemma 1. *$f(B)$ and $c(B)$ are polymatroid function.*

Proof. When a new role r is added to the set B, $N(B \cup \{r\})$ contains all the elements of $N(B)$, so $f(B)$ is increasing. For $f(B) = |N(B) \cap T|$, because $N(B) \cap T$ is a set, $f(B)$ can be seen as the number of elements in the set.

For any two set X and Y, $|X| + |Y| \geq |X \cup Y| + |X \cap Y|$. So $f(B)$ is submodular. When $B = \emptyset$, $N(B) \cap T = \emptyset$, then $f(\emptyset) = 0$. Hence, $f(B)$ is a polymatroid function.

Similarly, $c(B) = \sum_{p_i \in (N(B)-T)} \omega_i$ is the sum of elements' weight values in set $N(B) - T$, which is increasing and meets $c(\emptyset) = 0$. It is also submodular. Hence, f and c are polymatroid functions.

The least privilege problem is to find $\min c(B) : f(B) = f(R), B \subseteq R$ on 2^R, which is a minimum submodular cover with submodular cost, and the following result is true.

Theorem 1. *The least privilege problem is NP-hard.*

In user-role assignment, there are three sets: U denotes the set of users, R denotes the set of roles, and P denotes the set of privileges. Construct a graph $URA = (U \cup R \cup P, E_1 \cup E_2)$: if $u_i \in U$ contains role $r_j \in R$, there is an edge $(u_i, r_j) \in E_1$; if $r_j \in R$ contains privilege $p_l \in P$,there is an edge $(r_j, p_l) \in E_2$ (Fig. 3). Suppose $|U| = k$, $|R| = m$, $|P| = n$. Each user has different target privileges, so what need to do is to make sure that each user can get its target privilege with non-target privileges as little as possible.

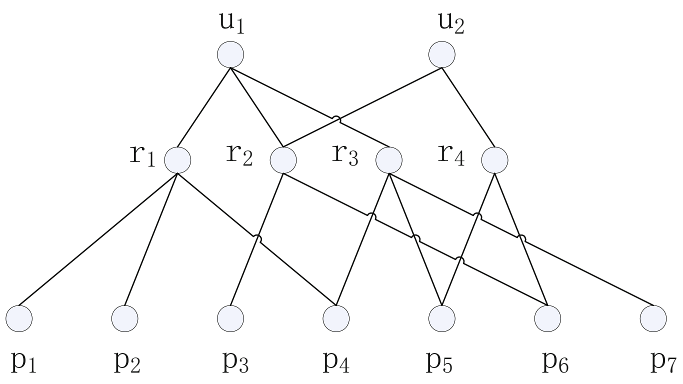


Fig. 3. The relations of users, roles and privileges

3 Greedy Algorithm

In this section, a greedy algorithm is introduced for presenting an approximation of the least privilege user-role assignment problem.

3.1 Parameter Definition

Let u_i denote the user with $1 \leq i \leq k$, and ω_i denote the weight for privilege p_i with $1 \leq i \leq n$. In this model, let T_i denote the set of target privileges for the user u_i . Let B_i denote the set of roles that are assigned to user u_i , and B_{all} is the total set of assigned roles, that is $B_{all} = \bigcup_i B_i$.

$\forall r \in R, \triangle_r f(B) = f(B \cup \{r\}) - f(B)$,which means the number of new target privileges brought via adding role r. When $\triangle_r f(B) = 0$, it implies adding role r is not necessary, because the new role cannot bring any new target privilege. The reason for this, may be the target privileges that role r contains are existing in the set of B, or there is no target privilege in role r at all.

Define the function for the subset B as follows: $e(B) = \sum_{p_i \in N(B) \cap T} \omega_i$, which means the sum of elements' weight values of target privileges in B, and $\forall r \in R, \triangle_r e(B) = e(B \cup \{r\}) - e(B)$.

In the case all the weights equal to 1, $e(B) = f(B)$ and $\triangle_r e(B) = \triangle_r f(B)$.

3.2 Algorithm

The greedy algorithm for user-role assignment is described in Algorithm 1. For each single user, the algorithm will be run once in order to complete its user-role assignment. In step 4, if some roles contain some target privileges which are not included in B_i, then the one which contains the largest ratio of marginal target privilege over redundant cost will be selected out (step 5) and added to B_i (step 6). The "while loop" will not stop until the specific user gets all its target privileges.

Algorithm 1. Greedy Algorithm for User-Role Assignment with Weight (GA-UAW)

```
1:  $B \leftarrow \emptyset$;
2:  $B_i \leftarrow \emptyset, 1 \leq i \leq k$;
3:  for $|T_i|>0, 1 \leq i \leq k$ do
4:      while $\exists r \in R, \triangle_r f(B_i)>0$ do
5:          select $r \in R$ with maximum $\frac{\triangle_r e(B_i)}{c(B_i \cup \{r\})}$;
6:          $B_i \leftarrow B_i \cup \{r\}$;
7:      end while
8:  end for
9:  output $B_i, 1 \leq i \leq k$;
10: output B;
```

3.3 Results

In the above algorithm, the privileges can be assigned to roles automatically. Similarly, the target privileges of users are generated automatically. Hence, this algorithm can adapt to all the systems. To measure the efficiency of the greedy algorithm, this paper uses an enumeration method to make comparison in both "time of running" and "the number of redundant privileges". To see the comparative result clearly, this paper gives simulation results of single-user case.

When $|U| = 1$ and $|P| = 50$, the running time by using the enumeration method is higher than one by using the greedy algorithm apparently as the number of roles increases (Table 1). $\omega = 1$ denotes that in the case all the weights equal to 1, and random ω denote the general case where the weights are assigned randomly. It is clear that the larger the number of roles is, the longer time that the enumeration method cost. However, the running time cost by the greedy algorithm almost unchanged.

Table 1. The running time comparison in case 1

The No. of roles	Time of running(s)		
$\|U\| = 1$	Enumeration	Greedy	
$\|P\| = 50$		$\omega = 1$	random ω
5	0.203	0.172	0.219
10	3.797	0.562	0.422
15	21.187	0.531	0.532
20	398.282	0.875	0.781
25	1582.47	1.375	0.985
30	4206.56	1.172	0.953
35	10381.9	1.516	1.156
40	20978.6	2.016	2

Table 2 shows the number of redundant privileges that are assigned to the user by both the enumeration method and the greedy algorithm in the case all the weights equal to 1. The enumeration method has listed all the user-role assignments including the optimal one. Compare with the optimal solution, the greedy algorithm can give a close solution in a shorter time. For a large company, the number of both users and roles are large, so the user-role assignment will cost a lot of time to get the best solution by using enumeration algorithm.

Table 2. The number of redundant privileges comparison in case 1

The No. of roles	The No. of redundant privileges	
$\|U\| = 1$ $\|P\| = 50$	Enumeration (optimal solution)	Greedy
5	26	26
10	24	25
15	27	27
20	19	20
25	14	16
30	20	21
35	17	18
40	20	20

3.4 Performance Analysis

Define the curvature of the submodular cost c to be $\rho = \min_{B:min-costcover} \frac{\sum_{r\in B} c(r)}{c(B)}$ [11], and $H(k) = 1 + \frac{1}{2} + ... + \frac{1}{k}$ is the k-th Harmonic number [7]. According to the previous work, if f is integer-valued, then the greedy solution of GSC is a $\rho H(\gamma)$ -approximation where $\gamma = \max_{r\in R} f(r)$ [11]. Hence, this algorithm is a $\rho H(\gamma)$-approximation. In this algorithm, the goal is to assign roles to users with least privilege principle. Since the user-role assignment process is the same for each user, the whole performance ratio can be shown by analyzing performance ratio of Algorithm 1. Let T denote the set of target privileges, and B is the set of roles that are assigned to the user by greedy algorithm.

Then $H(\gamma) = 1 + \frac{1}{2} + ... + \frac{1}{\gamma} = 1 + \frac{1}{2} + ... + \frac{1}{\max_{r\in R} f(r)} \le 1 + \ln \max_{r\in R} f(r) \le 1 + \ln |T|$. In the case all the weights equal to 1, $c(B) \le \sum_{r\in B} c(r)$. So $1 \le \rho = \min_{B:min-costcover} \frac{\sum_{r\in B} c(r)}{c(B)} \le \frac{|T|(|P| - |T|)}{|P| - |T|} = |T|$. That is $1 \le \rho \le |T|$. Hence, $H(\gamma) \le \rho H(\gamma) \le |T|(1 + \ln |T|)$.

In general case where the weights are assigned randomly, $c(B) \le \sum_{r\in B} c(r)$. Suppose $B = \{r_1, r_2, \cdots, r_s\}$ through the algorithm, and the target privilege set is $T = \{p_1, p_2, \cdots, p_{|T|}\}$. So $1 \le \rho = \min_{B:min-costcover} \frac{\sum_{r\in B} c(r)}{c(B)} \le \frac{|T|(\sum_{p_i\in(N(R)-T)} \omega_i)}{\sum_{p_i\in(N(R)-T)} \omega_i} = |T|$. That is $1 \le \rho \le |T|$. Therefore, $H(\gamma) \le \rho H(\gamma) \le |T|(1 + \ln |T|)$.

Hence, the best performance ratio is $H(\gamma)$, and the worst one is $|T|(1 + \ln |T|)$.

4 Conclusion

In this paper, the least privilege principle is proved to be NP-hard, and an approximation algorithm is given. Based on this algorithm, the user-role assignment can be efficiently done satisfying least privilege principle.

References

1. Barka, E., Sandhu, R.: Framework for role-based delegation models. In: Proceedings 16th Annual Computer Security Applications Conference (ACSAC 2000), pp. 168–176. IEEE Comput. Soc., Los Alamitos (2000)
2. Barka, E., Sandhu, R.: A role-based delegation model and some extensions. In: 23rd National Information Systems Security Conference (2000)
3. Chen, L., Crampton, J.: Inter-domain role mapping and least privilege. In: SACMAT 2007: Proceedings of the 12th ACM symposium on Access control models and technologies, pp. 157–162. ACM Press, New York (2007)
4. Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, R.: Proposed nist standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4(3), 224–274 (2001)
5. Ke Xue, S.T., Ge, L.: Least-privilege-based access control model for job execution in grid. In: ISDPE, pp. 301–303. IEEE Computer Society, Los Alamitos (2007)

6. Lai, C.: Quantitative enforcement of the principle of least privilege in rbac and an efficient fault tolerant cryptosystem. PhD thesis (2007)
7. Nemhauser, G., Wolsy, L.: Interger and Combinatorial Optimization. Wiley, Chichester (1999)
8. Schlegelmilch, J., Steffens, U.: Role mining with orca. In: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, pp. 168–176. ACM Press, New York (2005)
9. Vaidya, J., Atluri, V., Warner, J.: Roleminer: Mining roles using subset enumeration. In: CCS 2006: Proceedings of the 13th ACM Conference on Computer and Communications Security, New York, United States. Association for Computing Machinery, pp. 144–153 (2006)
10. Wainer, J., Kumar, A.: A fine-grained, controllable, user-to-user delegation method in rbac. In: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, pp. 59–66. ACM Press, New York (2005)
11. Wan, P.-J., Du, D.-Z., Pardalos, P., Wu, W.: Greedy approximations for minimum submodular cover with submodular cost. Computational Optimization and Applications 45(2), 463–474 (2010)
12. Xiaojun, M.: A boundary-based access control model for sensitive information. In: IFITA, pp. 685–689. IEEE Computer Society, Los Alamitos (2009)

Towards Minimum Delay Broadcasting and Multicasting in Multihop Wireless Networks

Maggie X. Cheng and Quanmin Ye

Department of Computer Science,
Missouri University of Science and Technology, Rolla, MO 65401
{chengm,qy4y4}@mst.edu

Abstract. End-to-end delay is defined as the total time it takes for a single packet to reach the destination. End-to-end delay, along with end-to-end throughput, is a determinant factor of the user-experienced data transmission time. It is an important QoS metric for both unicast and multicast applications. In this paper, we focus on the delay performance of multicast and broadcast applications.

In multihop wireless networks, end-to-end delay is a result of many factors including the length of a route (in hops) and the interference level of the links along the route. In fact, the sum of interference of links along a route is a good indicator of end-to-end delay. We propose a linear programming based routing scheme to achieve the minimum overall path interference. Through simulation, we show that the proposed routing scheme is better than the well-known shortest path tree based multicasting such as MOSPF.

1 Introduction

In multihop wireless networks with omnidirectional antennas, the signal from one transmitter could reach many unintended receivers and interfere with the reception of these neighbors. As a result, transmissions that are interfering with each other cannot be scheduled to happen at the same time. Throughput and delay both are affected by the interference in the environment, and they collectively decide the user-experienced data transmission time for file transferring.

End-to-end delay is the time it takes for a single packet to reach its destination. It is related to the number of hops on the routing path, and the interference level on each link along the path. We assume a TDMA based scheduling scheme is used at the MAC layer (The routing scheme proposed in this paper can work with any MAC layer protocol). Without interference from other flows, it is easy to see that a path of six hops introduces more delay than a path of four hops. As shown in Fig. 1(a) and (b), if the path S-a-b-c-d-e-D is taken, it takes 6 time slots to reach the destination; but if the path S-k-l-m-D is taken, it takes only 4 time slots to reach the destination. To see how interference adversely affects delay, we compare the end-to-end delay in Fig. 1(c) and (d). We assume there are 5 distinct slot numbers to use. If the FCFS (First-Come-First-Service) scheduling policy is used, and a packet is scheduled to use the next available slot as soon as it arrives, then it takes only 5 time slots to reach the destination in Fig. 1(c), but it takes 10 time slots to reach the destination in Fig. 1(d) because of the interference from the other flow. The numbers on links are slot numbers.

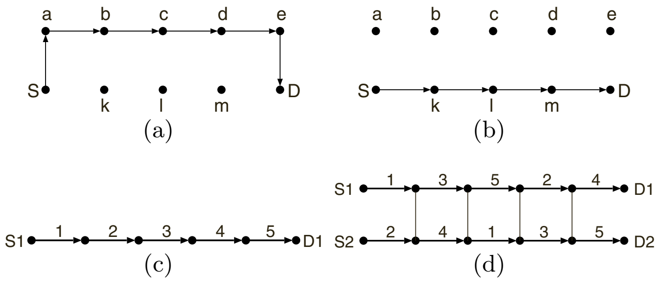


Fig. 1. Delay changes with path length and the interference level on the path

To reduce the end-to-end delay, optimization techniques can be applied at the network layer to achieve minimum-delay routing, or at the MAC layer to achieve minimum-delay transmission scheduling. Our previous work [1] has addressed the scheduling problem for unicast traffic in multihop wireless networks. In this paper, we address minimum-delay routing problem in multicasting and broadcasting applications. The scope of this paper is on network layer only.

Since both the number of hops and the interference level of links on the path can adversely affect end-to-end delay, the accumulated interference on a routing path is a good indicator of end-to-end delay. The actual delay won't be determined until a MAC layer scheduling scheme is given. So instead of directly minimizing delay, we aim to select the routing path that has the minimum accumulated interference. Our current work [2] showed that the path with smaller accumulated interference leads to smaller delay for unicast traffic, and we continue to use it for multicasting/broadcasting.

To accurately account for the effect of interference on end-to-end delay, one must consider how interference affects the underlying scheduling at the MAC layer. Transmissions that are conflicting with each other cannot happen simultaneously, therefore they must be scheduled one after another, and this is the main cause of delay. The number of time slots needed to schedule a group of links is equal to the maximum number of mutual conflicting links in the group. This is equivalent to find the maximum clique on a conflict graph [3]. However, to find the maximum clique is an NP-hard problem [4]. In this paper, we used an approximate linear constraint to replace the clique constraint and formulated the multicasting/broadcasting problems as integer linear programs. The linear constraint in this paper is tighter (more accurate) than the previously known linear constraint in the literature and yet sufficiently captures the conflicting relation among the links in a multihop wireless network.

The rest of the paper is organized as follows. In Section 2, we briefly survey the most related work in minimum delay multicast and broadcast in wireless networks; in Section 3 and Section 4, we present the mathematical optimization models and algorithms for a series of multicast/broadcast routing problems. We first address the single-source multicast problem, and then we extend it to multiple sources. For the multi-source multicasting problem, we first present the offline version, in which all sources and their destinations are given before the routing decision is made and then the online version, in which a routing decision has to be made before the next request arrives; in Section 5 we compare our algorithms with the well known MOSPF (Multicast Open Shortest Path First) multicast routing algorithm; in Section 6, we conclude the paper with discussion and future works.

2 Related Work

Delay optimization is often considered at the MAC layer through transmission scheduling or at the network layer through routing. Chatterjee et al. [5] presented when the routing tree is given for a sensor network how to determine the time slot of each node such that the maximum latency to send a packet from a node to the sink is minimized. Chaporkar et al. [6] addressed the MAC layer multicast problem as an instance of the stochastic shortest path problem and developed an optimal transmission strategy for minimum delay multicasting. Sarkar et al. [7] addressed the energy-delay tradeoff problem and formulated the problem as a constrained optimization problem that achieves the minimum energy while satisfying the constraint on average packet delay. Pereira et al. [8] addressed delay optimization problem for a random access MAC protocol. They presented an accurate analytical model to derive the optimal transmission probability of each mobile node to minimize delay.

At the network layer, Sivrikaya et al. [9] presented an algorithm to compute the minimum-delay path for networks with STDMA. Wan et al. [10] presented approximation algorithms for minimum latency aggregation in sensor networks, which is to compute an aggregation tree for sensor nodes so that the makespan of the aggregation schedule is the minimum. Li et al. [11] studied how to select the routing path with the minimum end-to-end delay in multi-radio wireless mesh networks and developed routing protocols for both single-channel and multi-channel wireless mesh networks. Alzahrani and Woodward [12] proposed a localized QoS routing algorithm and addressed delay optimization problem.

In addition to the stand-alone network layer and MAC layer solutions, cross-layer joint design and optimization has become a solution to achieve the overall optimal network performance. Cui et al. [13] crossed the network layer, the MAC layer and the physical layer to optimize routing, TDMA slot assignment, MQAM modulation rate and power on each link in order to minimize the worst-case packet delay. The cross-layer optimization problems are approximated by convex optimization problems and efficiently solved. Xia et al. [14] used a fuzzy logic system to consider the joint design of the physical layer, the data link layer and the application layer, and is proved to be effective in improving QoS and energy efficiency. Pakdehi et al. [15] introduced cross layer design between the

MAC layer and the physical layer to optimize the overall system throughput while preserving packet average delay time. Xiao [16] investigated joint design of network-coding and channel-coding and optimized delay performance through the tradeoff design between the network layer and the physical layer. Wang and Shroff [17] addressed the design of network codes and associated flow in network coding in a cross-layer design paradigm, and implemented a distributed algorithm for joint scheduling and rate-control. Ukil [18] proposed a cross-layer framework for WiMAX networks to optimize the system performance as well as maintaining the end-to-end QoS of individual users, and presented the cross-layer resource allocation and scheduling scheme in the WiMAX system.

While the aforementioned work addressed multi-user cross-layer optimization, Fu et al. [19] investigated solving the cross-layer optimization across time from a single users perspective. To consider the dynamic nature of user traffic, efficient online algorithm was proposed to solve it.

The proposed work largely depends on the understanding of the impact of interference on end-to-end delay. Most previous work on interference modeling has been used to address throughput optimization (e.g. [3, 20–24]). Jain et al. [3] first used conflict graphs to model the effect of wireless interference under a simplified protocol model; Qiu et al. [21] continued to use conflict graphs to model interference under IEEE 802.11 interference model; Further in [22], Qiu et al. proposed a physical interference model, which is based on measured interference rather than distance between nodes. In addition to interference modeling, Padhye et al. [25] focused on the estimation of interference and studied the effect of interference on aggregated network throughput based on IEEE 802.11 model. In our previous work [24], we proposed an interference model for directed graphs based on a different MAC layer transmission scheme.

3 Broadcasting and Multicasting with Single Source

The paper is about broadcasting/multicasting among a group of wireless nodes. Since each node has limited transmission range, multihop paths can be used to connect two nodes. We first deal with broadcasting. The requirement for broadcasting is that the source-to-destination delay is as small as possible, and meanwhile, the receivers receive the information at roughly the same time. Since we cannot guarantee that they receive at the same time, we want to minimize the variation in receiving time.

3.1 Network-Wide Broadcasting

For a network of $|V|$ nodes, one node is the source node, and all others are destinations. The task is to compute a broadcasting tree rooted at the source node. Since it is a network-wide broadcasting, all nodes are included in the tree. The internal nodes of the tree are both forwarding nodes and destination nodes.

Let l_{ij} be the decision variable. $l_{ij} = 1$ indicates link (i, j) (from i to j) is on the tree; $l_{ij} = 0$ otherwise. $l_{ij,v}$ is a decision variable specific to each destination.

$l_{ij,v} = 1$ indicates link (i,j) is on the path to reach destination node v. $l_{ij} = 1$ as long as link (i,j) is on the routing path to one of the destinations. So the broadcast tree is a collection of links $\{(i,j) : l_{ij} = 1\}$.

Let I_v denote the accumulated interference along the routing path from the source to destination v. Since the actual delay won't be determined until the MAC layer scheduling algorithm is given, we use the accumulated interference on the path as an indicator of delay. We wish to minimize the sum of the accumulated interference on the routing paths to all destinations, therefore the objective function is: to minimize $\sum_{v \in V-\{s\}} I_v$.

For some applications, in addition to reduce overall delay, it also requires that the earliest receiving time and the latest receiving time be as close as possible. This requirement is important when an action is expected upon receiving the message. We wish that all receiving nodes take action at roughly the same time. To this end, we introduce a weight factor α, a constant between 0 and 1, and then the objective function is revised to:

Minimize

$$\alpha \sum_{v \in V-\{s\}} I_v + (1-\alpha)(maxI - minI) \tag{1}$$

The first term is to minimize total delay, the second term is to minimize the difference between the maximum delay and minimum delay, hence to minimize delay variation.

The given wireless network is represented as $G = (V, E)$. In the following, node s is the source node, and $s \in V$. In order to facilitate the use of flow conservation property, we create a node t as the virtual sink node. t is not a real node in the wireless network, so $t \notin V$. We add an edge from each node $v \in V - \{s\}$ to node t; but when we consider interference, we exclude the edges $\{(v,t)\}$. Let N_i denote the neighbors of node i. N_i could possibly include the virtual sink t.

The constraints for the optimization problem are:

Subject to

$$\sum_{j \in N_s} l_{sj,v} = 1, \quad \forall v \in V - \{s\} \tag{2a}$$

$$\sum_{j \in V} l_{tj} = 0 \tag{2b}$$

$$\sum_{i \in N_j} l_{ij} = 1, \quad \forall j \in V - \{s\} \tag{2c}$$

$$\sum_{j \in N_i} l_{ij,v} - l_{ji,v} = 0, \forall i \in V - \{s\}, \forall v \in V - \{s\} \tag{2d}$$

$$I_v = \sum_{(i,j) \in E} I_{ij} \cdot l_{ij,v}, \quad \forall v \in V - \{s\} \tag{2e}$$

$$I_{ij} = l_{ij} + \sum_{l \in N_i, l \neq j,t} l_{il} + \sum_{k \in N_j, k \neq i,t} l_{jk} + \sum_{(k,l) \in N2_{ij}, k,l \neq t} l_{kl}, \quad \forall (i,j), i,j \neq t \tag{2f}$$

$$I_v \leq maxI, \quad \forall v \in V - \{s\} \tag{2g}$$

$$I_v \geq minI, \quad \forall v \in V - \{s\} \tag{2h}$$

$$l_{ij} \geq \sum_{v \in V - \{s\}} l_{ij,v}/C, \quad \forall \texttt{link}(i,j) \tag{2i}$$

$$l_{ij} = \{0,1\} \quad \forall \texttt{link}(i,j) \tag{2j}$$

$$l_{ij,v} = \{0,1\} \quad \forall \texttt{link}(i,j), \forall v \in V - \{s\} \tag{2k}$$

(2a) and (2d) specify the flow conservation property for each flow from the source to each destination v; (2b) specifies the property of the virtual sink node t, i.e., no flow comes out from t; (2c) specifies the tree property so that each node except the source has exactly one parent node; (2e) defines the accumulated interference on the path to destination v; (2g) and (2h) define the maximum and minimum interference among all paths; (2i) is to make sure that link (i,j) is selected into the tree as long as one of the routing path chooses link (i,j).

(2f) is the key to model the interfering relationship in wireless networks. We assume at the link layer, each data packet is acknowledged by the receiving node, so all links within two hops of each other interfere with each other. This is the IEEE 802.11 interference model and has been widely used in the research community. In order to map the amount of interference to the MAC layer delay resulting from a TDMA scheduling algorithm, we need to consider the maximum number of mutually conflicting links of each link, because links mutually conflicting with each other cannot be scheduled to use the same slot, therefore the total number of mutually conflicting links represent the number of slots needed to schedule the group of links. However, this requires to find the maximum clique on the conflict graph. To avoid solving the NP-hard clique problem, we replace the constraint on a clique with (2f). It can be proven that (2f) sufficiently captures the mutual conflict relationship among links. If each link needs one slot, then the RHS of (2e) indicates the sufficient number of slots needed to schedule the links on the routing path.

Since both I_{ij} and $l_{ij,v}$ are variables, (2e) is not linear, so the above program is not an integer linear program yet. We can use an iterative procedure to convert it to an integer linear program. Let k be the index of iterations. In the beginning, we initialize $k = 1$, and initial interference $I_{ij}^0 = 1$ for all links. At the k^{th} iteration, we do the following:

Step 1: In (2e), we use the value of I_{ij}^{k-1} from the $k-1^{th}$ iteration to replace variable I_{ij}. The above nonlinear program becomes an integer linear program (ILP);

Step 2: Solve the ILP using the LP-rounding based scheme to get routing information $l_{ij,v}$ and l_{ij}. Let $\overline{I}_{ij}$ denote the value of variable I_{ij}. Update $\overline{I}_{ij}$ based on the new routing information;

Step 3: Set $I_{ij}^k = \overline{I}_{ij}$;

Step 4: Update I_v using the new routing information $l_{ij,v}$ and new interference value I_{ij}^k, and calculate the objective value;

Step 5: $k = k + 1$.

Repeat steps 1-5 until the objective value converges or oscillates within a threshold value.

Remarks:

1. In the beginning, we set $I_{ij}^0 = 1$ for all links, so the result is a shortest path tree using interference as link weights.
2. It oscillates sometimes because we are using interference, a dynamic traffic load indicator, as link weight. Oscillation is unavoidable just like in the Link State algorithm. Recall that when we use traffic load as the metric, the Link State routing algorithm oscillates ([26]).
3. To avoid or reduce oscillation, we can desynchronize routing as follows:
   – At each iteration, only allow half (or other percentage of preference) of the routing paths to update, and the other half use the current routes. Let $I_{v,1/2}$ denote the median of all I_v's. Select half of the destinations $V_{0.5} \subset V - \{s\}$ such that $\forall v \in V_{0.5}$, $I_v \leq I_{v,1/2}$. Before we enter the next iteration, we plug in the the current values for $l_{ij,v}$ for all $v \in V_{0.5}$ and use them as constants. It is shown by experiments that the algorithm converges faster this way.

3.2 Multicasting

For multicasting, we use D_s to denote the group of destinations for source s. From broadcasting to multicasting, a difficulty rises because some nodes may not be included in the tree at all. The traditional methods that directly build a spanning tree then prune for the multicast group will not lead to the optimal solution. Instead, a mathematical model that directly minimizes the objective function and guarantees a connected path from the source to each destination will work much better.

The following is the optimization model for multicasting. At this point, we can view broadcasting as a special case of multicasting, where the destination group includes all nodes in $V - \{s\}$. A notable difference from the broadcasting model to the multicasting model is from (2c) to (4c) and (4d). In (2c), each node except the source has exactly one parent node; in (4c), each destination node has exactly one parent node and all other nodes have at most one parent node. The flow conservation property guarantees that there is a connected path to reach each destination node $v \in D_s$.

Minimize

$$\alpha \sum_{v \in D_s} I_v + (1-\alpha)(maxI - minI) \tag{3}$$

Subject to

$$\sum_{j \in N_s} l_{sj,v} = 1, \quad \forall v \in D_s \tag{4a}$$

$$\sum_{j \in V} l_{tj} = 0 \tag{4b}$$

$$\sum_{i \in N_j} l_{ij} = 1, \quad \forall j \in D_s \tag{4c}$$

$$\sum_{i \in N_j} l_{ij} \leq 1, \quad \forall j \in V - \{s\} - D_s \tag{4d}$$

$$\sum_{j \in N_i} l_{ij,v} - l_{ji,v} = 0, \forall i \in V - \{s\}, \forall v \in D_s \tag{4e}$$

$$I_v = \sum_{(i,j) \in E} I_{ij} \cdot l_{ij,v}, \quad \forall v \in D_s \tag{4f}$$

$$I_{ij} = l_{ij} + \sum_{l \in N_i, l \neq j,t} l_{il} + \sum_{k \in N_j, k \neq i,t} l_{jk} + \sum_{(k,l) \in N2_{ij}, k,l \neq t} l_{kl}, \quad \forall (i,j), i,j \neq t \tag{4g}$$

$$I_v \leq maxI, \quad \forall v \in D_s \tag{4h}$$

$$I_v \geq minI, \quad \forall v \in D_s \tag{4i}$$

$$l_{ij} \geq \sum_{v \in D_s} l_{ij,v}/C, \quad \forall \texttt{link}(i,j) \tag{4j}$$

$$l_{ij} = \{0,1\} \quad \forall \texttt{link}(i,j) \tag{4k}$$

$$l_{ij,v} = \{0,1\} \quad \forall \texttt{link}(i,j), \forall v \in D_s \tag{4l}$$

4 Multicasting with Multiple Sources

In the previous section, we addressed the multicast routing problem with a single source node. A multicast session can be described as a tuple (s, D_s). In this section, we address the optimal routing for multiple sessions. The optimization obejective is the overall delay, or the difference between the latest receiving time and the earliest receiving time, or any tradeoff between the two objectives. We first present the offline version algorithm, and then we present the online version algorithm.

4.1 Offline Algorithm

In the offline version, all multicast sessions are given before the routing decision is made. We know the set of sources S, and the destination nodes D_s for each source $s \in S$. To compute the optimal solution for multiple sessions, the routing decision variables are defined with one more dimension— the first dimension s indicates it is for source s. The interference variable I_{ij} denotes the total interference on link (i,j) including interference from all sources. $I_{s,v}$ denotes the accumulated interference from source s to destination v.

Minimize

$$\sum_{s\in S}\left(\alpha\sum_{v\in D_s} I_{s,v} + (1-\alpha)(maxI_s - minI_s)\right) \tag{5}$$

to

$$\sum_{j\in N_s} l_{s,sj,v} = 1, \quad \forall s\in S,\ \forall v\in D_s \tag{6a}$$

$$\sum_{j\in V} l_{s,tj} = 0, \quad \forall s\in S \tag{6b}$$

$$\sum_{i\in N_j} l_{s,ij} = 1, \quad \forall s\in S,\ \forall j\in D_s \tag{6c}$$

$$\sum_{i\in N_j} l_{s,ij} \le 1, \quad \forall s\in S,\ \forall j\in V-\{s\}-D_s \tag{6d}$$

$$\sum_{j\in N_i}(l_{s,ij,v} - l_{s,ji,v}) = 0, \forall s\in S, \forall i\in V-\{s\}, \forall v\in D_s \tag{6e}$$

$$I_{s,v} = \sum_{(i,j)\in E} I_{ij}\cdot l_{s,ij,v},\ \forall s\in S, \forall v\in D_s \tag{6f}$$

$$I_{ij} = \sum_{s\in S}\left(l_{s,ij} + \sum_{l\in N_i, l\neq j,t} l_{s,il} + \sum_{k\in N_j, k\neq i,t} l_{s,jk} + \sum_{(k,l)\in N2_{ij}, k,l\neq t} l_{s,kl}\right), \quad \forall (i,j), i,j\neq t \tag{6g}$$

$$I_{s,v} \le maxI_s, \quad \forall s\in S, \forall v\in D_s \tag{6h}$$

$$I_{s,v} \ge minI_s, \quad \forall s\in S, \forall v\in D_s \tag{6i}$$

$$l_{s,ij} \ge \sum_{v\in D_s} l_{s,ij,v}/C, \quad \forall s\in S,\ \forall \mathtt{link}(i,j) \tag{6j}$$

$$l_{s,ij} = \{0,1\} \quad \forall s\in S,\ \forall \mathtt{link}(i,j) \tag{6k}$$

$$l_{s,ij,v} = \{0,1\} \quad \forall s\in S,\ \forall \mathtt{link}(i,j), \forall v\in D_s \tag{6l}$$

In this formulation, (6a)–(6e) indicate that flow conservation is preserved in each session from the source to each destination; and (6g) defines the total interference on link (i,j) from all flows. The above nonlinear program can be solved by first converting to an ILP using the iterative method described in section 3 and LP-rounding based method to solve the ILP. As the number of concurrent sessions increases, the computation time increases. However, the number of concurrent sessions in reality is a small number.

4.2 Online Algorithm

In the online version, routing decision for a multicast session must be made before the next request is received. Once the routing decision is made, it cannot be changed later. But since the joining of new flows, interference on links may be changed later, and therefore what is considered as optimal before may not be optimal after the new session starts. We wish to find the online routing solution that is as close to the offline optimal solution as possible.

Let s' denote the source node for the current session, and $D_{s'}$ denote the destination group for s'. The following mathematical optimization program solves the routing problem for the current request with the objective of optimizing the overall delay of all sessions. (8a)-(8e) specify that flow conservation must be satisfied from s' to each destination node $v \in D_{s'}$. (8f)-(8i) update interference for all sessions, not just for the current source s'. Note that in (8f) and (8g), the routing information for the current source s' is unknown, but for all other sources $s \in S - \{s'\}$ the routing information is known.

Minimize

$$\sum_{s \in S} \left(\alpha \sum_{v \in D_s} I_{s,v} + (1-\alpha)(maxI_s - minI_s) \right) \tag{7}$$

Subject to

$$\sum_{j \in N_{s'}} l_{s',s'j,v} = 1, \quad \forall v \in D_{s'} \tag{8a}$$

$$\sum_{j \in V} l_{s',tj} = 0 \tag{8b}$$

$$\sum_{i \in N_j} l_{s',ij} = 1, \quad \forall j \in D_{s'} \tag{8c}$$

$$\sum_{i \in N_j} l_{s',ij} \le 1, \quad \forall j \in V - \{s'\} - D_{s'} \tag{8d}$$

$$\sum_{j \in N_i} (l_{s',ij,v} - l_{s',ji,v}) = 0, \forall i \in V - \{s'\}, \forall v \in D_{s'} \tag{8e}$$

$$I_{s,v} = \sum_{(i,j) \in E} I_{ij} \cdot l_{s,ij,v}, \ \forall s \in S, \forall v \in D_s \tag{8f}$$

$$I_{ij} = \sum_{s \in S} \left(l_{s,ij} + \sum_{l \in N_i, l \ne j,t} l_{s,il} + \sum_{k \in N_j, k \ne i,t} l_{s,jk} + \sum_{(k,l) \in N2_{ij}, k,l \ne t} l_{s,kl} \right), \quad \forall (i,j), i,j \ne t \tag{8g}$$

$$I_{s,v} \le maxI_s, \quad \forall s \in S, \forall v \in D_s \tag{8h}$$

$$I_{s,v} \ge minI_s, \quad \forall s \in S, \forall v \in D_s \tag{8i}$$

$$l_{s',ij} \ge \sum_{v \in D_{s'}} l_{s',ij,v}/C, \quad \forall \texttt{link}(i,j) \tag{8j}$$

$$l_{s',ij} = \{0,1\} \quad \forall \texttt{link}(i,j) \tag{8k}$$

$$l_{s',ij,v} = \{0,1\} \quad \forall \texttt{link}(i,j), \forall v \in D_{s'} \tag{8l}$$

The program defined by (7)-(8l) is still nonlinear due to the nonlinear constraint in (8f). For the first request, we solve it in the same way as in the single source multicasting in section 3. For subsequent requests, the interference caused by previous requests is used as input and added to the total interference, so the problem remains to solve becomes the same problem as a single source multicasting problem, and the iterative algorithm presented in section 3 is applied again.

5 Simulation

In the following simulations, we choose a wireless network of 30 nodes, randomly deployed in a region of 1500×1500. Transmission range is properly set to make sure the network is connected. The source node is randomly chosen. For broadcast, all other nodes are destinations; for multicast, 5 nodes are randomly chosen to be the destination nodes.

5.1 Broadcasting and Multicasting with a Single Source

We first compare the proposed method with the MOSPF method for single source broadcasting and multicasting. MOSPF uses the shortest path (in hops) from the source to each destination, so the multicast tree is the superposition of shortest paths originated from a single source.

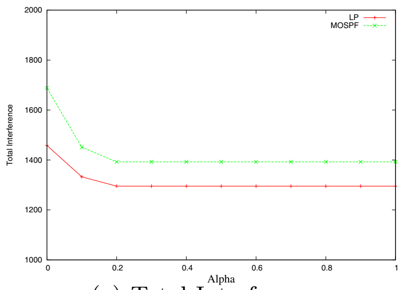
We compare our method with MOSPF on the total interference and the difference between the most interfered path and the least interfered path. The results in Fig. 2 show that the proposed method achieves smaller total interference with smaller difference between the largest value and the smallest value, which implies the overall delay is smaller, and the gap between the first receiving time and the last receiving time is also smaller than in MOSPF.

5.2 Multicasting with Multiple Sources

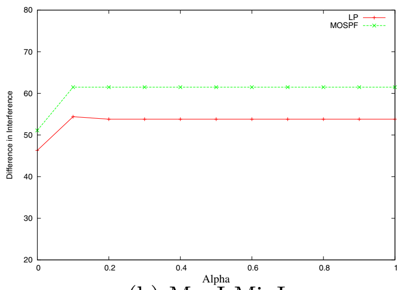
In the second simulation, we extend to multicasting with multiple sources. We first test the offline version algorithm based on the model in Section 4.1, then we test the online version based on the model in Section 4.2. Both are compared with MOSPF.

Offline Algorithm. In the offline version, MOSPF uses hop counts as routing metric. The proposed LP-offline algorithm uses the model in Section 4.1 to compute the routing solution. Fig. 3 shows the total interference and interference difference as α changes from 0 to 1.

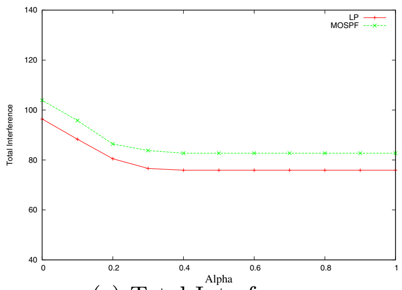
Online Algorithm. The online version MOSPF would choose the shortest path for each destination in the current session. The proposed online algorithm would choose the best routes based on the optimization model in Section 4.2. There are 5 multicast sessions. In each multicast session, 5 destination nodes are randomly chosen from the network. Fig. 4 shows the total interference and interference difference in the order of arrival.



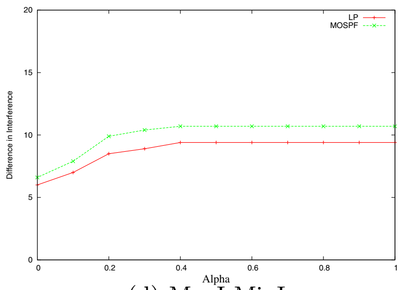
(a) Total Interference



(b) MaxI-MinI

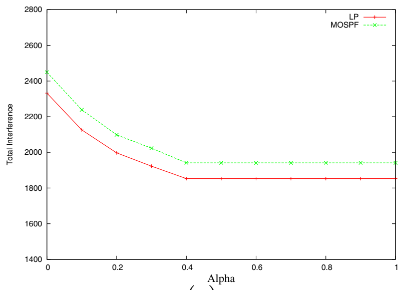


(c) Total Interference

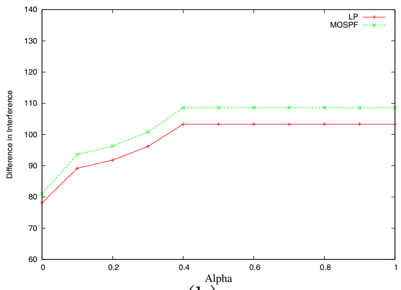


(d) MaxI-MinI

Fig. 2. With one source node, (a) and (b)Network-wide broadcasting, (c) and (d) Multicasting



(a)



(b)

Fig. 3. Offline algorithms for multiple sessions, (a) Total Interference, (b) Interference Difference (Max-Min)

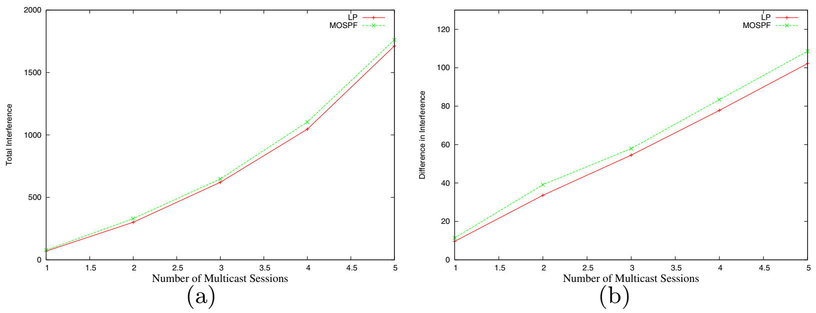


Fig. 4. Online algorithms for multiple sessions, (a) Total Interference, (b) Interference Difference (Max-Min)

6 Conclusion

End-to-end delay, along with end-to-end throughput, is a determinant factor of the user-experienced data transmission time. It is an important QoS metric for both unicast and multicast applications. In this paper, we focus on the delay performance of multicast and broadcast applications. We address the question of how to minimize overall delay and variation of delay among multicast receivers in multihop wireless networks. We first relate the end-to-end delay to the interference on the routing path, and then use mathematical optimization to compute the routing solution for multicast. To effectively estimate the impact of wireless interference on delay, we propose to use a sufficient condition in place of the NP-hard clique condition in the integer linear program, developed offline and online algorithms to address multisession multicast communication problem. Through simulation, we show that the proposed routing scheme is better than the well-known shortest path tree based multicasting algorithm such as MOSPF.

Acknowledgement. Maggie Cheng and Quanmin Ye are supported in part by National Science Foundation under grant CNS-0841388.

References

1. Cheng, M.X., et al.: Link activity scheduling for minimum end-to-end latency in multihop wireless sensor networks (2011) manuscript
2. Cheng, M.X., et al.: Minimum delay routing in multihop wireless networks (2011) manuscript
3. Jain, K., Padhye, J., Padmanabhan, V.N., Qiu, L.: Impact of interference on multihop wireless network performance. In: MobiCom 2003: Proceedings of the 9th annual international conference on Mobile computing and networking, pp. 66–80. ACM, New York (2003)
4. Garey, M., Johnson, D.: Computers and Intractability, A Guide to the Theory of NP-Completeness. FREEMAN, New York (1974)

5. Chatterjee, P., Das, N.: A cross-layer distributed tdma scheduling for data gathering with minimum latency in wireless sensor networks. In: Wireless VITAE 2009, pp. 813–817 (May 2009)
6. Chaporkar, P., Sarkar, S.: Minimizing delay in loss-tolerant mac layer multicast. In: Third International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks. WIOPT 2005, pp. 358–367 (April 2005)
7. Sarkar, M., Cruz, R.L.: A mac layer power management scheme for efficient energy delay tradeoff in a wlan. The International Journal of Computer and Telecommunications Networking 51, 1–6 (2007)
8. Pereira, M., Bernardo, L., Dinis, R., Oliveira, R., Carvalho, P., Pinto, P.: Delay optimization on a p-persistent mac protocol for a multi-packet detection in sc-fde system. In: Wireless Communications and Networking Conference (WCNC), pp. 1–6. IEEE, Los Alamitos (April 2010)
9. Sivrikaya, F., Yener, B.: Minimum delay routing for wireless networks with stdma. Wireless Networks 15(6), 755–772 (2007)
10. Wan, P.J., Huang, S.C.H., Wang, L., Wan, Z., Jia, X.: Minimum-latency aggregation scheduling in multihop wireless networks. In: MobiHoc 2009, pp. 185–194 (2009)
11. Li, H., Cheng, Y., Zhou, C., Zhuang, W.: Minimizing end-to-end delay: A novel routing metric for multi-radio wireless mesh networks. In: INFOCOM 2009, pp. 46–54. IEEE, Los Alamitos (2009)
12. Alzahrani, A., Woodward, M.: End-to-end delay in localized qos routing. In:11th IEEE Singapore International Conference on Communication Systems, ICCS 2008, pp. 1700–1706 (November 2008)
13. Cui, S., Madan, R., Goldsmith, A., Lall, S.: Cross-layer energy and delay optimization in small-scale sensor networks. IEEE Transactions on Wireless Communications 6(10), 3688–3699 (2007)
14. Xia, X., Ren, Q., Liang, Q.: Cross-layer design for mobile ad hoc networks: energy, throughput and delay-aware approach. In: Wireless Communications and Networking Conference, WCNC 2006, vol. 2, pp. 770–775. IEEE, Los Alamitos (2006)
15. Pakdehi, A., Ashtiani, F.: Cross-layer optimization of adaptive modulation and coding preserving packet average delay time. In: IEEE GLOBECOM 2008, December 4, pp. 1–5 (2008)
16. Xiao, M.: Cross-layer design of rateless random network codes for delay optimization. In: IEEE International Conference on Communications ICC 2010, pp. 1–6 (May 2010)
17. Wang, C.C., Shroff, N.: On wireless network scheduling with intersession network coding. In: 42nd Annual Conference on Information Sciences and Systems, CISS 2008, pp. 30–35 (March 2008)
18. Ukil, A.: Cross-layer optimization in qos aware next generation wireless networks. In: 7th International Conference on Information, Communications and Signal Processing, ICICS 2009, pp. 1–5 (December 2009)
19. Fu, F., van der Schaar, M.: Cross-layer optimization with complete and incomplete knowledge for delay-sensitive applications. In: 17th International Packet Video Workshop, PV 2009, pp. 1–10 (May 2009)
20. Rangwala, S., Gummadi, R., Govindan, R., Psounis, K.: Interference-aware fair rate control in wireless sensor networks. In: SIGCOMM 2006: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 63–74. ACM, New York (2006)

21. Li, Y., Qiu, L., Zhang, Y., Mahajan, R., Zhong, Z., Deshpande, G., Rozner, E.: Effects of interference on wireless mesh networks: Pathologies and a preliminary solution. In: HotNets 2007 (November 2007)
22. Qiu, L., Zhang, Y., Wang, F., Han, M.K., Mahajan, R.: A general model of wireless interference. In: MobiCom 2007: Proceedings of the 13th annual ACM international conference on Mobile computing and networking, pp. 171–182. ACM, New York (2007)
23. Li, Y., Qiu, L., Zhang, Y., Mahajan, R., Rozner, E.: Predictable performance optimization for wireless networks. In: SIGCOMM 2008: Proceedings of the ACM SIGCOMM 2008 conference on Data communication, pp. 413–426. ACM, New York (2008)
24. Cheng, M., Gong, X., Cai, L.: Link rate allocation under bandwidth and energy constraints in sensor networks. In: Global Telecommunications Conference, IEEE GLOBECOM 2008, December 4-30, pp. 1–5. IEEE, Los Alamitos (2008)
25. Padhye, J., Agarwal, S., Padmanabhan, V.N., Qiu, L., Rao, A., Zill, B.: Estimation of link interference in static multi-hop wireless networks. In: IMC 2005: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement, pp. 28–28. USENIX Association, Berkeley (2005)
26. Kurose, J.F., Ross, K.W.: Computer Networking: A Top-Down Approach. Addison-Wesley, Reading (2010)

Author Index