Der-Tsai Lee
Danny Z. Chen
Shi Ying (Eds.)

LNCS 6213

Frontiers in Algorithmics

4th International Workshop, FAW 2010
Wuhan, China, August 2010
Proceedings

Springer

Lecture Notes in Computer Science 6213

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

Der-Tsai Lee Danny Z. Chen Shi Ying (Eds.)

Frontiers in Algorithmics

4th International Workshop, FAW 2010
Wuhan, China, August 11-13, 2010
Proceedings

Springer

Volume Editors

Der-Tsai Lee
Academia Sinica, Institute of Information Science 20, Section 2
128 Academia Road, Nankang, Taipei 11529, Taiwan
E-mail: dtlee@ieee.org

Danny Z. Chen
University of Notre Dame, Department of Computer Science and Engineering
Notre Dame, IN 46556, USA
E-mail: dchen@cse.nd.edu

Shi Ying
Wuhan University, State Key Laboratory of Software Engineering
Wuhan, Hubei 430072, China
E-mail: yingshi@whu.edu.cn

Library of Congress Control Number: 2010930688

CR Subject Classification (1998): F.2, D.2, F.1, G.2, F.3, G.2.2

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

ISSN 0302-9743
ISBN-10 3-642-14552-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-14552-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The papers in this volume were selected for presentation at the 4th International Frontiers of Algorithmics Workshop (FAW 2010), held during August 11-13, 2010 at Wuhan University, Wuhan, China. Previous meetings of this workshop were held in Lanzhou (2007), Changsha (2008), and Hefei (2009).

In response to the Call-for-Papers, 57 extended abstracts were submitted from 12 countries and regions, of which 28 were accepted. The submitted papers were from China, France, Hong Kong, India, Israel, Italy, Japan, Republic of Korea, Lebanon, The Netherlands, Taiwan, and USA.

The papers were evaluated by an international Program Committee. Each paper was evaluated by at least three Program Committee members, with possible assistance of the external referees, as indicated by the referee list found in these proceedings. In addition to the selected papers, the workshop also included three invited presentations by Mikhail J. Atallah, Deyi Li, and Kurt Mehlhorn, and two training sessions by John Hopcroft and Xiaotie Deng for providing students and young researchers with advanced research experience.

We thank all Program Committee members and the external referees for their excellent work, especially given the demanding time constraints. Furthermore, we thank the General Conference Co-chairs John Hopcroft and Deyi Li, and the Steering Committee Co-chairs Xiaotie Deng and Franco Preparata. It has been a wonderful experience to work with all of them. We also thank the three invited speakers, the two training session speakers, and all authors who submitted papers for consideration. They all contributed to the high quality of the workshop.

Finally, we thank all local organizers, led by Rong Peng, and the colleagues of the State Key Lab of Software Engineering and the School of Computer, Wuhan University, who worked tirelessly to put in place the logistical arrangements of the workshop and to create and maintain the website of the workshop. It was their hard work that made the workshop possible and enjoyable.

May 2010

Der-Tsai Lee
Danny Z. Chen
Shi Ying

Conference Organization

General Conference Co-chairs

John E. Hopcroft
Deyi Li

Steering Committee Co-chairs

Xiaotie Deng
Franco P. Preparata

Program Chairs

Der-Tsai Lee (Chair)
Danny Z. Chen (Co-chair)
Shi Ying (Co-chair)

Program Committee

Nancy Amato
Tetsuo Asano
Franz Aurenhammer
Gill Barequet
Amitabh Chaudhary
Ning Chen
Siu-Wing Cheng
Francis Chin
Kyung-Yong Chwa
Ovidiu Daescu
Peter Eades
Qizhi FANG
Rudolf Fleischer
Subir Ghosh
Sun-Yuan Hsieh
Ming-Yang Kao
Naoki Katoh
Lian Li
Angsheng Li
Xuemin Lin
Zhiyong Liu
Chi-Jen Lu
Pinyan Lu
Shuang Luan
Martin Noellenburg
Peter Rossmanith
Michiel Smid
Takeshi Tokuyama
Lusheng Wang
Haitao Wang
Feng Wang
Sue Whitesides
Xiaodong Wu
Jinhui Xu
Jinyun Xue
Jianping Yin
Guochuan Zhang
Louxin Zhang
Jian Zhang
Xiao Zhou
Hong Zhu
Binhai Zhu

Local Organization

Rong Peng (Chair)
Feng Wang
Ying Zheng
Mao Ye

External Reviewers

Muhammad Cheema
Yam K. Cheung
Hu Ding
Leah Epstein
Jywe-Fei Fang
Andreas Gemsa
Heng Guo
Xin Han
Xia Hua
Takehiro Ito
Justie Su-tzu Juan
Mong-Jen Kao
Bastian Katz
Joachim Kneis
Lap Kei Lee
Cheng-Chung Li
Fei Li
Jian Li
Qun Liu
Yunlong Liu
Wei Mi
Sang-il Oum
Yicheng Pan
Sheng-Lung Peng
Branislav Stojkovic
Kei Uchizawa
Dorothea Wagner
Kejun Wang
Ting Wang
Bang Ye Wu
Chenggang Wu
Yongan Wu
Zhilin Wu
Mingji Xia
Wei Xiong
Lei Xu
Deshi Ye
Hung-I Yu
Teng-Kai Yu
Honghai Zhang
Wenjie Zhang
Yongding Zhu
Marko Zivanic

Table of Contents

Progress on Certifying Algorithms

Kurt Mehlhorn and Pascal Schweitzer

Max-Planck-Institut für Informatik, Saarbrücken, Germany

A *certifying algorithm* is an algorithm that produces with each output, a *certificate* or witness (easy-to-verify proof) that the particular output has not been compromised by a bug. A user of a certifying program P (= the implementation of a certifying algorithm) inputs x, receives an output y and a certificate w, and then checks, either manually or by use of a checking program, that w proves that y is a correct output for input x. In this way, he/she can be sure of the correctness of the output without having to trust P. We refer the reader to the recent survey paper [9] for a detailed discussion of certifying algorithms.

1 An Example

We illustrate the concept by an example. A *matching* in a graph G is a subset M of the edges of G such that no two share an endpoint. A matching has maximum cardinality if its cardinality is at least as large as that of any other matching. Figure 1 shows a graph and a maximum cardinality matching. Observe that the matching leaves two nodes unmatched, which gives rise to the question whether there exists a matching of larger cardinality. What is a witness for a matching being of maximum cardinality? Edmonds in his seminal papers [1,2] on how to compute maximum matchings in polynomial time introduced the following certificate: An *odd-set cover* OSC of G is a labeling of the nodes of G with nonnegative integers such that every edge of G is either incident to a node labeled 1 or connects two nodes labeled with the same number $i \geq 2$.

Theorem 1 ([1]). *Let N be any matching in G and let OSC be an odd-set cover of G. For any $i \geq 0$, let n_i be the number of nodes labeled i. Then*

$$|N| \leq n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor.$$

Proof. For i, $i \geq 2$, let N_i be the edges in N that connect two nodes labeled i and let N_1 be the remaining edges in N. Then

$$|N_i| \leq \lfloor n_i/2 \rfloor \text{ for } i \geq 2 \quad \text{and} \quad |N_1| \leq n_1$$

and the bound follows.

It can be shown (but this is non-trivial) that for any maximum cardinality matching M there is an odd-set cover OSC with

$$|M| = n_1 + \sum_{i \geq 2} \lfloor n_i/2 \rfloor, \tag{1}$$

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 1–5, 2010.

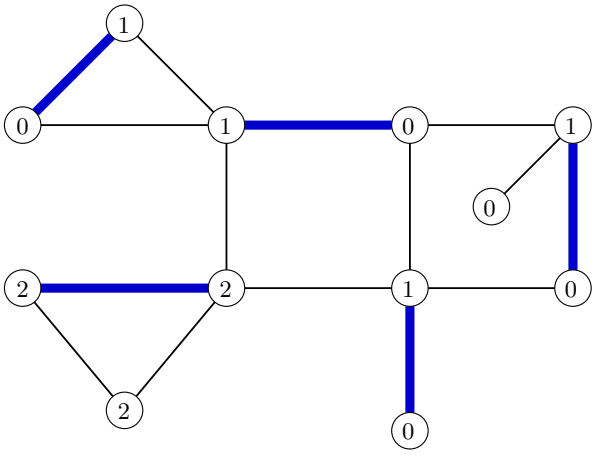


Fig. 1. The node labels certify that the indicated matching is of maximum cardinality: All edges of the graph have either both endpoints labelled as two or at least one endpoint labelled as one. Therefore, any matching can use at most one edge with two endpoints labelled two and at most four edges that have an endpoint labelled one. Therefore, no matching has more than five edges. The matching shown consists of five edges.

thus certifying the optimality of M. In such a cover all n_i with $i \geq 2$ are odd, hence the name.

A *certifying algorithm for maximum cardinality matching* returns a matching M and an odd-set cover OSC such that (1) holds. Edmonds [1,2] gave the first efficient algorithm for maximum cardinality matchings. The algorithm is certifying and outputs a matching M and an odd-set cover OSC satisfying (1). By the argument above, the odd-set cover proves the optimality of the matching. Observe, that is it *not* necessary to understand why odd-set covers proving optimality always exist. It is only required to understand the simple proof of Theorem 1, showing that equation (1) proves optimality. Also, a correct checking program which controls whether a set of edges is a matching and a node labelling is an odd-set cover which together satisfy (1) is easy to write.

2 Formal Verification

We repeat the last two sentences of the introduction. It is only required to understand the simple proof of Theorem 1, showing that equation (1) proves optimality. Also, a correct program which checks whether a set M of edges is a matching and a node labelling OSC is an odd-set cover which together satisfy (1) is easy to write. Such a program would have to verify that M is a subset of E, that the degree of every vertex with respect to M is at most one, that the node labelling OSC is an odd-set cover, and that (1) holds.

If the preceding paragraph holds true, we should be able to substantiate it in two ways:

1. Turn the proof of Theorem 1 into a formal proof.
2. Prove the correctness of the checking program.

In ongoing work of the first author with Eyad Alkassar, Christine Rizkallah, and Norbert Schirmer we have done exactly this. We formalized and proved Theorem 1 using Isabelle [7] and we wrote the checker in Isabelle and C and proved it correct using Isabelle and VCC [14], respectively. We plan to extend this work to a large fraction of the certifying algorithms covered in [9].

3 Three-Connectivity of Graphs

An undirected graph is 3-connected if the removal of any two vertices does not disconnect it. Linear time algorithms for this problem are known [6,10], but none of them are certifying. The fastest certifying algorithms have quadratic running time $O(n^2)$, see [12,9]. We review the algorithm given in the latter paper.

Certifying that a graph is not 3-connected is simple; it suffices to provide a set S of vertices, $|S| \leq 2$, such that $G \setminus S$ is not connected. The non-certifying algorithms [6,10] above compute such a set if the input is not 3-connected. However, if the input is 3-connected, the only output returned is "input is 3-connected". Gutwenger et. al. [5] implemented the algorithm of [6] and report that it incorrectly declares some non-3-connected graphs 3-connected. They provide a correction.

Tutte [13] introduced a certificate for 3-connectedness, a sequence of edge contractions resulting in the K_4, the complete graph on 4 vertices. We call an edge e of a 3-connected graph G *contractible* if the contracted graph G/e, (i.e. the graph obtained by replacing the end-vertices of e by a single vertex neighboring all vertices previously adjacent to one of the endpoints) is 3-connected. A separating pair is a pair of vertices whose removal disconnects the graph.

Lemma 1. *Let $e = (x, y)$ be an edge of a simple graph G whose end-vertices have a degree of at least 3. If G/e is 3-connected, then G is 3-connected.*

Proof. Since contractions cannot connect a disconnected graph, the original graph G is connected. There are no cut-vertices in G, as they would map to cut-vertices in G/e.

Any separating pair of G must contain one of the end-vertices of edge e. Otherwise the pair is also separating in G/e. It cannot contain both x and y, otherwise the contracted vertex xy is a cut-vertex in G/e. Suffices now to show that x, u, with $u \in V(G) \setminus y$ is not a separating pair. Suppose otherwise, then the graph $G - \{x, u\}$ is disconnected, but the graph $G\{x, y, u\}$ is not. Thus $\{y\}$ is a component of $G - \{x, u\}$. But this is a contradiction since y has degree at least 3 in G.

To certify the 3-connectivity of a graph G, it thus suffices to provide a sequence of edges which, when contracted in that order, have endpoints with a degree of a least 3 and whose contraction results in a K_4. We call such a sequence a *Tutte sequence*. We now focus on how to find the contraction sequence, given a 3-connected graph.

The $O(n^2)$ algorithm needs three ingredients: First we require the $O(n^2)$ algorithm by Nagamochi and Ibaraki [11] that finds a sparse spanning 3-connected subgraph of G with at most $3n-6$ edges. Second we require a linear time algorithm for 2-connectivity. Third we require a structure theorem, that shows how to determine a small candidate set of edges among which we find a contractible edge.

Theorem 2 (Krisell [8]). *If no edge incident to a vertex v of a 3-connected graph G is contractible, then v has a least four neighbors of degree 3, which each are incident with two contractible edges.*

Consider now a vertex v of minimal degree in a 3-connected graph. If it has degree three, it cannot have four neighbors of degree three and hence must have an incident contractible edge. If it has degree four or more, it cannot have a neighbor of degree three (because otherwise, its degree would not be minimal) and hence must have an incident contractible edge. Also note that an edge xy in a 3-connected graph is contractible, if $G-\{x,y\}$ is 2-connected.

We explain how to find the first $n/2$ contractions in time $O(n^2)$. By repeating the procedure we obtain an algorithm that has overall a running time of $O(n^2)$.

First use the algorithm by Nagamochi and Ibaraki [11]. The resulting graph has $3n-6$ edges. Thus while performing the first $n/2$ contractions, there will always be a vertex with degree at most $2\cdot 2\cdot 3 = 12$. Choosing a vertex of minimal degree, we obtain a set of at most 12 candidate edges, one of which must be contractible. To test whether an edge xy is contractible, we check whether $G-\{x,y\}$ is 2-connected with some linear time algorithm for 2-connectivity.

Theorem 3 ([12]). *A Tutte sequence for a 3-connected graph can be found in time $O(n^2)$.*

It remains a challenge to find a linear time certifying algorithm for 3-connectivity of graphs. A linear time certifying algorithm for graphs that contain a Hamiltonian cycle was recently found [3]; this assumes that the Hamiltonian cycle is part of the input. It was also shown recently [4] that for any DFS-tree T of a 3-connected graph G, there is a Tutte sequence that contracts only edges in T.

References

1. Edmonds, J.: Maximum matching and a polyhedron with 0,1 - vertices. Journal of Research of the National Bureau of Standards 69B, 125–130 (1965)
2. Edmonds, J.: Paths, trees, and flowers. Canadian Journal on Mathematics, 449–467 (1965)

3. Elmasry, A., Mehlhorn, K., Schmidt, J.M.: A Linear Time Certifying Triconnectivity Algorithm for Hamiltonian Graphs. Available at the authors home pages (March 2010)
4. Elmasry, A., Mehlhorn, K., Schmidt, J.M.: Every DFS-Tree of a 3-Connected Graph Contains a Contractible Edge. Available at the authors home pages (February 2010)
5. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
6. Hopcroft, J.E., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM Journal of Computing 2(3), 135–158 (1973)
7. Isabelle theorem prover, `http://isabelle.in.tum.de/`
8. Kriesell, M.: A survey on contractible edges in graphs of a prescribed vertex connectivity. Graphs and Combinatorics, 1–33 (2002)
9. Mehlhorn, K., McConnell, R., Näher, S., Schweitzer, P.: Certifying Algorithms. Available at the first author's homepage and submitted for publication
10. Miller, G.L., Ramachandran, V.: A new graph triconnectivity algorithm and its parallelization. Combinatorica 12(1), 53–76 (1992)
11. Nagamochi, H., Ibaraki, T.: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica 7, 583–596 (1992)
12. Schmidt, J.M.: Construction sequences and certifying 3-connectedness. In: 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010), Nancy, France (2010), `http://page.mi.fu-berlin.de/jeschmid/pub`
13. Tutte, W.: A theory of 3-connected graphs. Indag. Math. 23, 441–455 (1961)
14. VCC, a mechanical verifier for concurrent C programs, `http://vcc.codeplex.com/`

Computational Geometry for Uncertain Data
(Abstract of Keynote Talk)

Mikhail J. Atallah

Department of Computer Science, Purdue University
mja@cs.purdue.edu

Abstract. The talk will review recent results and algorithmic challenges for computational geometry problems in the context of uncertain data. This is an active area of investigation in the database community, and we introduce it through the specifics of the maximal elements problem (called the skyline problem in the database community): Rather than being a point, an uncertain object is a set of points called instances, each with an associated probability; instances of the same uncertain object can be geometrically far from each other, and are mutually exclusive (i.e., at most one of them can occur). For this version of the maximal elements problem, the input is a collection of m uncertain objects, whose total number of instances is n, and the problem is to compute for each of these n instances the probability that it is a maximal point, i.e., that it occurs for its own object and is not dominated by any occurring instance from another object.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, p. 6, 2010.

On Foundations of Services Interoperation in Cloud Computing

Deyi Li[1], Haisu Zhang[2], Yuchao Liu[1,3], and Guishen Chen[1]

[1] Institute of Electronic System Engineering,
Beijing, 100141, China
[2] College of Command Automation, PLA University of Science and Technology,
Nanjing, 210007, China
[3] Department of Computer Science and Technology, Tsinghua University,
Beijing, 100084, China
leedeyi@tsinghua.edu.cn, zhanghaisu@139.com,
yuchao_liu@163.com, cgs@mail.tsinghua.edu.cn

Detailed Abstract. With a long-run accumulation of IT technologies, cloud computing becomes a new revolution after PC revolution in 1980s, the Internet revolution in 1990s, and the mobile Internet revolution in 2000s. Cloud computing is a paradigm of Internet computing, which takes software as a service oriented to a number of users with changing requirements from time to time, at the same time, takes the response of a requirement as an up-to-date best effort rather than a unique precise one. It is changing the way we share data, information and knowledge.

Services support direct reusing of application programs instead of software deployment, and improve usability of resource sharing. Clouds can be regarded as an enabler for interoperation of large scale service provisioning. Therefore, assembling of software became services aggregation under the mature understanding of interoperability.

Services interoperation may happen between a user (group) and a service, or among services. Users or their groups, services or their aggregations are all called agents here. The description of interoperability of agents focuses on three issues: their role, goal, and combination of services. The role issue characterizes the organization, roles, and actors of an agent, and describes the interaction and cooperation among them. The goal issue depicts the decomposition of goals and determines the constraint relationship among goals. While the two issues compose the problem domain, the third issue is related to the solution domain, in which process distinguishes atomic processes, and composite processes, and defines the input/output together with precondition/effect of processes respectively, the service guides the construction of service chains and aggregation of resources. Usually, the description in problem domain is qualitative, while the specification in solution domain is quantitative.

When services description faces anywhere-access and massive demands from its huge amount of users, described by natural languages with different scales, the capabilities of interoperation such as semantic and granular computing with uncertainty are essential.

These capabilities are related to a qualitative and quantitative transform model. More than ten years ago, we suggested a cloud model by means of

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 7–10, 2010.

second order normal distribution, to transform between quantitative data and qualitative concepts. Approximately speaking, normal distributions with two parameters: expectation (*Ex*) and entropy (*En*) are universally existent in statistics of preferential common requirements, collective interaction behavior and services aggregation process, etc. Over the Internet, if the above phenomenon are determined by the sum of a quantity of equal and independent statistical factors, and if every factor has minor influence, then the phenomenon obtain normal distribution approximately. However, there exist many occasions in which there are some dependent factors with great influence. The precondition of a normal distribution is no longer existed. To solve this problem, we propose a new parameter of hyper entropy (*He*) in order to relieve the precondition.

In cloud model, the overall property of cloud drops can be represented by three numerical characters: the expected value *Ex*, the entropy *En,* and the hyper-entropy *He*, corresponding to a qualitative concept. *Ex* is the mathematical expectation of the cloud drops of a concept. In other words, the cloud drop located at the *Ex* point is the most representative of the qualitative concept. *En* is the granular measurement of the qualitative concept, which connects both the randomness and the fuzziness of the concept. *He* is the uncertainty measurement of the entropy, i.e. the entropy of the entropy, showing to what degree a number of cloud drops become a common concept. That is to say, if *He* is greater than *En*, the concept cannot be formed any longer. The uncertainty of the concept can be represented by multiple numerical characters. In general, people get used to perform reasoning with the help of languages, rather than excessive mathematic calculation. So, it is adequate to employ these three numerical characters to reflect common concepts.

Forward Cloud Generator (FCG) algorithm transforms a qualitative concept with *Ex*, *En* and *He* into a number of cloud drops representing the quantitative description of the concept. While Reverse Cloud Generator (RCG) algorithm transforms a number of cloud drops into three numerical characters (*Ex*, *En, He*) representing the concept.

Generally speaking, a cloud generator is used to form a special concept for a group of data only. Different groups of data in the whole distribution domain may lead to many different concepts by using Gaussian Mixture Model (GMM). Any quantitative data will belong to a certain concept. Furthermore, fewer qualitative concepts with larger granular can be further extracted by Cloud Synthesis Method (CSM). These concepts may be the foundation of the ontology construction as semantic interoperation.

The CSM process is given in this way: choose two clouds with the shortest distance on *Ex*, and merge them into a new cloud if and only if *He* of the new cloud is small than *En*. The new cloud can be generated by RCG using the data covered by the original two clouds. The process ceases until no two clouds can be merged.

Based on cloud model, GMM and CSM, the gap between qualitative description of requirements on the problem domain and quantitative services on solution domain can be bridged.

Usually, the interoperation of services is multi-attributive, such as functionality, reliability, usability, efficiency, maintainability and portability etc. Functionality includes suitability, accuracy, interoperability and security;

Reliability includes maturity, fault tolerance and recoverability; Usability includes understandability, learn-ability and operability; Efficiency includes time behavior and resource utilization; Maintainability includes analyzability, changeability, stability and testability; at last, portability includes adaptability, install-ability, coexistence and replace-ability. All of all, service interoperation must be comparable to the qualitative requirements on the problem domain described by natural languages. At the same time, the service interoperation has to search and match a specific service or service aggregation with quantitative description from a service pool on the solution domain, as if pick up drops from cloud model.

As an example for understanding the above bridging method, the time behavior measurements of a service, which is quantitative on the solution domain, are going to be transformed to qualitative concepts on the problem domain. The test dataset comes from seekda (http://webservices.seekda.com/). Seekda's Web Services portal provides a search platform for public direct access to web services, which can enable users to find web services based on a catalogue of more than 28,000 service descriptions. Services listed at seekda cover a wide range of functionality in map, weather, sports, shopping and entertainment etc., and can be integrated into more capacious services. At present seekda verifies if a service is up once a day, and reports a measurement of availability by means of the frequency whether the server correctly implements the SOAP protocol daily.

Suppose the frequencies of 536 services returned from seekda for analysis, Fig. 1 shows the frequency of availability distribution. Let $f(x)$ be the frequency distribution function of the availability. With the help of cloud model and GMM, the qualitative concepts forming process is formulated as:

$$f(x) \to \sum_{i=1}^{n} a_i * C_i(Ex_i, En_i, He_i)$$

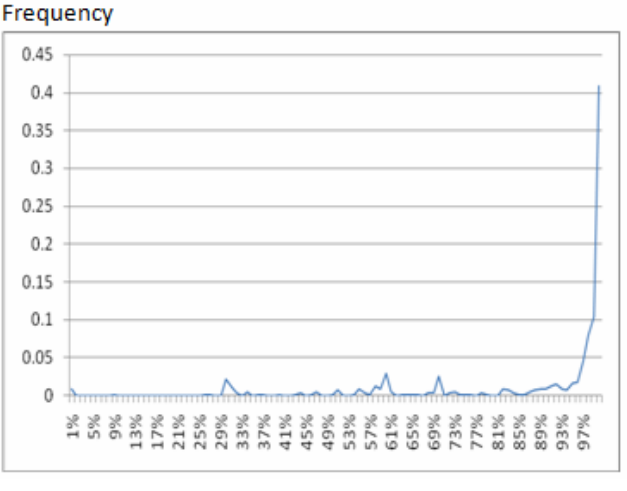
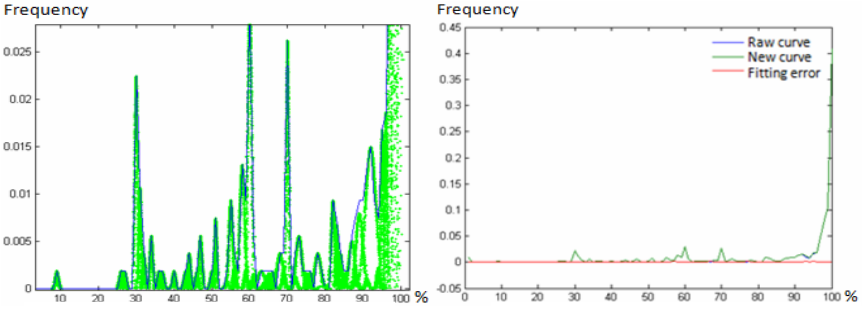


Fig. 1. Frequency of availability distribution

where a_i is the magnitude coefficient, and n is the number of discrete concepts after transformation. Fig.2(a) shows that 53 clouds C_i (Ex_i, En_i, He_i) (i=1,…,n, n=53) fits the frequency curve, and the fitting error in Fig.2(b) is less than the one asked for. For further extraction, CSM is used to extract out 8 clouds representing concepts with larger granular, as shown in Fig. 3. From these results, we can further rank and recommend services via the qualitative concepts, such as services with high performance are those covered by the cloud whose Ex is about 99% or 100%, and the services with the availability below 95% will not be recommend as the ones with high performance.



(a) 53 cloud models fitting the distribution by GMM

(b) fitting error between cloud models and distribution

Fig. 2. Qualitative concepts described by cloud models

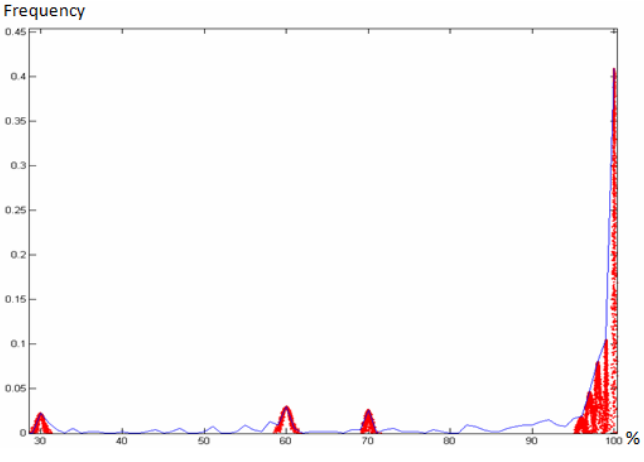


Fig. 3. Main concepts extracting by CSM (8 clouds)

For mature understanding of services interoperability, the cloud model, GMM and CSM may form a foundation of ontology, and bridge the gap between qualitative description of requirements on the problem domain and quantitative services on solution domain. For the future work, the long term monitoring of services need to maintain the concepts extraction system automatically.

Mechanism Design for Multi-slot Ads Auction in Sponsored Search Markets

Xiaotie Deng[1], Yang Sun[1], Ming Yin[2], and Yunhong Zhou[3]

[1] Department of Computer Science, City University of Hong Kong
deng@cs.cityu.edu.hk, yangsun7@student.cityu.edu.hk
[2] School of Software, Tsinghua University, Beijing, China
yinm07@mails.tsinghua.edu.cn
[3] Baidu Research, Beijing, China
zhouyunhong@baidu.com

Abstract. In this paper, we study pricing models for multi-slot advertisements, where advertisers can bid to place links to their sales webpages at one or multiple slots on a webpage, called the multi-slot AD auction problem. We develop and analyze several important mechanisms, including the VCG mechanism for multi-slot ads auction, the optimal social welfare solution, as well as two weighted GSP[1]-like protocols (mixed and hybrid). Furthermore, we consider that forward-looking Nash equilibrium and prove its existence in the weighted GSP-like pricing protocols.

We prove properties and compare revenue of those different pricing models via analysis and simulation.

Keywords: Sponsored Search, Mechanism Design, multi-slot AD auction, GSP, Forward-looking Nash Equilibrium.

1 Introduction

Targeted advertising with search results has been the major factor for successful commercialization of search engines, with billions of dollars in profits[1]. Those ADs regularly contain a headline, a line of ad copy, and a link to the target address.[7] They appear wherever Internet users may drift away from the original webpage's subject matter to look at and potentially click on those ADs to place an order to the commercial goods they desire. Search engines charge the advertisers according to both their bids and their click through rates (CTR). The Generalized Second Price (GSP) protocol, first developed by Google and widely adopted across the industry, has been studied most extensively in the academic field.

With the boost of internet bandwidth in recent years, rich ADs, that include images or videos, have become a new possibility in on-line advertisement. With rich ADs, user can interact with ads and learn more about a product without having to leave the webpage they are currently on so that the advertising effects of ADs are clearly better than text ADs.

[1] Generalized Second Price Auction.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 11–22, 2010.

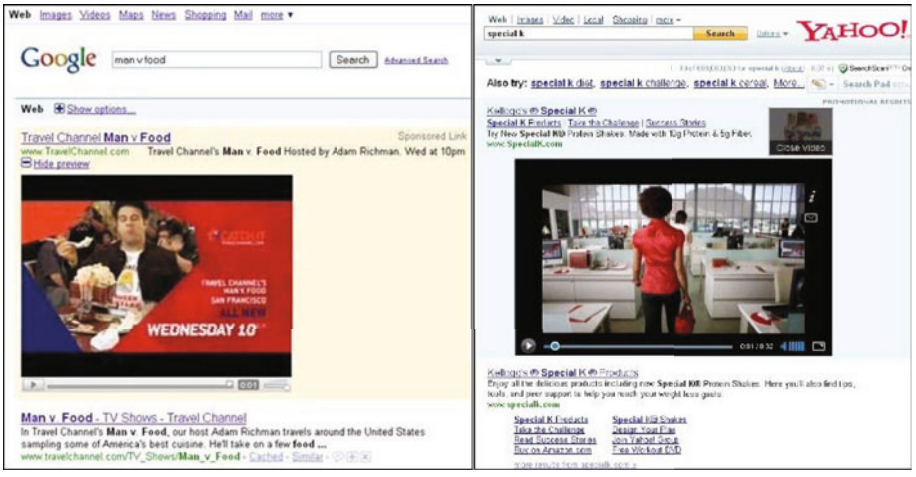


Fig. 1. Result pages with rich ads from Google and Yahoo! respectively

With both Google and Yahoo! it is possible for the advertisers to add videos and images to their ads. It was reported that click-through rates rises up to 25 percents for rich ads with videos and images.[11] More and more rich ads have appeared in online advertisements.

Clearly, it is an important research direction to understand pricing models and mechanisms for rich ads. In this work, we develop a simple model for this problem, then make an initial step to study various known protocols originated from the traditional single slot markets, discuss their strength and weakness, and test their profitability through analysis and simulation. We organize the presentation as follows. We introduce the multi-slot pricing problem in section 2. In section 3, we discuss several solutions and protocols, including the OPT solution and the VCG solution, two weighted GSP-like pricing protocols, as well as the forward-looking Nash equilibrium in the weighted GSP-like protocol. In section 4, we make a full comparison of expected revenue among mechanisms including weighted GSP-like, Optimal, VCG and that under the forward-looking Nash equilibrium via simulation. Finally, we conclude our work in section 5.

2 Rich ADs Auction as a Mechanism Design Problem

In selling an advertising space to all the advertisers(bidders) with single-slot ads or multi-slot ads, the seller's input is described in the following parameters:

Input. The input of the algorithm contains a list of bids and a list of slots, and each bid is characterized by $\theta_i(v_i, b_i, \beta_i, m_i) \in \Theta$, where v_i is the bidder's private value for each click, and m_i denotes the number of continuous slots its advertisement occupies, we assume $m_i \in \{1, n\}$ and $n \in \mathbb{N}^+$. For concreteness,

let us assume $m_i \in \{1, 3\}$[2] in this paper. The seller also assigns a weight β_i to each bid θ_i, which is often added as a relevance or quality metric of the bidder's ad. If one bids b_i, her score is denoted by $s_i = b_i \cdot \beta_i$.

Data Structure and notations

a) $T : \{\theta_{t_1}, \theta_{t_2}, \cdots, \theta_{t_p}\}$: List of all single-slot bids sorted in decreasing order of $s_i \cdot \beta_i$.
b) $M : \{\theta_{m_1}, \theta_{m_2}, \cdots, \theta_{m_q}\}$: List of all multi-slot bids in decreasing order of $s_i \cdot \beta_i$. For simplicity, we assume that m_i=3 for all multi-slot bids in this paper.
c) $\mathbb{K} : \{\alpha_1, \alpha_2, \cdots, \alpha_K\}$: an ordered list of K slots with position factors: $\alpha_1 > \alpha_2 > \cdots > \alpha_K$,$\alpha_k$ which are the probabilities that a user will click an ad with $\beta_i = 1$ on the corresponding slots.

CTRs are assumed to be related to the position factor α_k, as what we get from statistics. Besides, they are also related to the weight β_i of the bidder which occupies the slot. We assume that the CTR of $\theta_i(v_i, b_i, \beta_i, m_i)$ in slots starting at α_k is:

$$ctr_{\alpha_k}^{m_i} = \beta_i \cdot \sum_{k}^{k+m_i-1} \alpha_k \tag{1}$$

3 Pricing Protocols

The advertising market requires a mechanism that takes in bids from the advertisers, and decides on allocations of ads to slots as well as prices of slots for assigned bidders to pay.

3.1 The Optimal (OPT) Solution

Firstly, we will introduce an allocation and payment rule which social welfare and revenue is optimal in this problem. This is the ultimate social optimal solution when the seller has the complete information all bidders' private values.

Allocation Rule. In this solution, the K advertising slots are allocated to bidders in descending orders of their scores. We follow a greedy allocation rule that from the top slot in $\mathbb{K}$, each step we choose a bid θ_i with the maximum s_i to fill the slot(s). The allocation rule is described as Algorithm 1.

Theorem 1. *The allocation in Algorithm 1 guarantees the social welfare to be the maximum, when all bidders bid truthfully.*

Proof. When all bidders bid truthfully, we regard b_i as v_i for each bidder. For a single slot α_k, if it is a single-slot bid θ_{t_i} which occupies it, the part this slot contributes to the social welfare is $\alpha_k \cdot \beta_i \cdot b_i$; if it is a multi-slot θ_{m_j} which

[2] This corresponds to a commonly used constant in contextual advertising where text ads occupy one ad slot thus $m_i = 1$, multimedia ads occupy 3 slots thus $m_i = 3$.

Algorithm 1. OPT Allocation Rule

Data: $\mathbb{K}\{\alpha_1, \alpha_2, \alpha_3 \cdots, \alpha_k\}$, $\mathrm{T}\{\theta_{t_1}, \theta_{t_2}, \cdots, \theta_{t_q}\}$, $\mathrm{M}\{\theta_{m_1}, \theta_{m_2}, \cdots \theta_{m_r}\}$
Result: An ordered list$< \theta >$ $\Re$ of allocation for K slots
begin
  for $i \leftarrow 1$ **to** K **do**
    $s_t \leftarrow Max(b_j \cdot \beta_j \mid \theta_j \in T), s_m \leftarrow Max(b_j \cdot \beta_j \mid \theta_j \in M)$
    if *i>=k-2 or (i<k-2 and* $s_t > s_m$*)* **then**
      Insert θ_t to List $\Re_0$
      Remove θ_t from T
    else
      Insert θ_m to List $\Re_0$
      Remove θ_m from M
      $i \leftarrow i + 2$
  if *at most 2* $\theta_t \in \Re_0$ **then**
    return $\Re_0$
  else
    List $\Re^{'} \leftarrow \Re_0$
    Remove last 3 θ_t in $\Re'$
    Insert $Max(b_j \cdot \beta_j \mid \theta_j \in M)$ to $\Re'$
    if $\sum_{\Re_0} v_i \cdot \beta_i \cdot \alpha_i > \sum_{\Re'} v_i \cdot \beta_i \cdot \alpha_i$ **then**
      return $\Re_0$
    else
      return $\Re^{'}$
end

occupies it, the part that the bid contributes to the social welfare is $(\alpha_k + \alpha_{k+1} + \alpha_{k+2}) \cdot \beta_j \cdot b_j$, and the part the slot contributes to the social welfare could be regarded as $\alpha_k \cdot \beta_j \cdot b_j$. In this way, all the multi-slot bids in the auction could be converted to three single-slot bids, and the bidding price of all three single-slot bids equal to that of the original multi-slot bid.

Assume that after following Algorithm 1 to allocate all the slots, the ordered queue we get is $\Re\{\theta_1, \theta_2, \cdots, \theta_n\}$. If the social welfare of it is not the largest among all possible queues, then we assume that the bid queue which gets the largest social welfare is $\Re'\{\theta'_1, \theta'_2, \cdots, \theta'_n\}$. In $\Re$ and $\Re'$, we have at least a pair of bids, say θ_i and θ_j, that θ_i ranks higher than θ_j in $\Re$, which indicates $b_i \cdot \beta_i > b_j \cdot \beta_j$, but ranks lower than θ_j in $\Re'$. Formally, $\Re\{\theta_1, \cdots, \theta_i, \cdots, \theta_j, \cdots, \theta_n\}, \Re'\{\theta'_1, \cdots, \theta_j, \theta'_{j+1}, \cdots, \theta'_{i-1}, \theta_i, \cdots, \theta'_n\}$ and $\sum_{\Re'} b_i \cdot ctr_i > \sum_{\Re} b_i \cdot ctr_i$.

If we switch θ_i and θ_j in $\Re'$, we will get a new queue $\Re''\{\theta'_1, \cdots, \theta_i, \theta'_{j+1}, \cdots, \theta'_{i-1}, \theta_j, \cdots, \theta'_n\}$. If both θ_i and θ_j are multi-slot bids or both are single-slot bids, then obviously we have:

$$\sum_{\Re''} b_i \cdot ctr_i - \sum_{\Re'} b_j \cdot ctr_j = (b_i\beta_i - b_j\beta_j) \sum \alpha_i + (b_j\beta_j - b_i\beta_i) \sum \alpha_j$$
$$= (b_i\beta_i - b_j\beta_j)(\sum \alpha_i - \sum \alpha_j) > 0$$

$\sum_{\Re''} b_i \cdot ctr_i > \sum_{\Re'} b_j \cdot ctr_j$, which contradicts the assumption that $\Re'$ receives the maximal social welfare among all possible queues.

If θ_i and θ_j are different types of bids, say θ_j is a single-slot bid while θ_i is a multi-slot one. We first convert θ_i into three single-slot bids θ_{i1}, θ_{i2} and θ_{i3}. Then, we could switch θ_i and θ_j through a group of switches: $\theta_j \Leftrightarrow \theta_{i3}, \theta_{i2} \Leftrightarrow \theta_{i1} \Leftrightarrow \cdots \Leftrightarrow \theta'_{j+1}, \theta_{i1} \Leftrightarrow \theta'_{i-1} \Leftrightarrow \cdots \Leftrightarrow \theta'_{j+1}$. As we know from the analysis above, the social welfare increases after each switch, so we still have $\sum_{\Re''} b_i \cdot ctr_i > \sum_{\Re'} b_j \cdot ctr_j$, which contradicts the assumption that $\Re'$ receives the maximal social welfare among all possible queues.

Similarly, we could prove that if θ_i is a multi-slot bid while θ_j is a single-slot one, $\sum_{\Re''} b_i \cdot ctr_i$ is still larger than that of $\Re'$, which leads to a contradiction.

Therefore, OPT Mechanism receives the maximal social welfare among all possible mechanisms.

Pricing Rule. The optimal solution forms a base for mechanism designs for our problem. Arguably, when the bidders participate in the market repeatedly for many rounds, there is a possibility that the seller may gradually learn the true values of the bidders. In such an idea case, we may charge each bidder its true value of the clicks. In this case we can achieve the optimal revenue for the seller.

$$p_{OPT}^{\theta_i,\alpha_k,m_i} = b_i \tag{2}$$

3.2 VCG Mechanism

Next, we will introduce another allocation and payment rule which social welfare is optimal for this problem. It is an instance of VCG auction in the case of multi-slot ads auction.

Allocation Rule. In the VCG mechanism, the K advertising slots are allocated to advertisers in descending orders of their scores, which is the same as the OPT allocation rule.

Pricing Rule. According to the definition of the VCG mechanism, the expected payment of θ_i should be calculated using the Clark's externality payment rule:

$$Pay_{VCG}^{\theta_i,\alpha_k,m_i} = Revenue_{OPT}^{\Theta\setminus\theta_i,\mathbb{K}} - Revenue_{OPT}^{\Theta\setminus\theta_i,\mathbb{K}\setminus\sum_{\alpha_k}^{\alpha_{k+m_i-1}}} \tag{3}$$

The former term denotes social value the search engine gets when θ_i is removed from the auction, the latter term is social value the search engine gets when θ_i and its occupied slots are removed from the auction, and hence the difference between them is the total harm caused by θ_i.

In our problem, $Pay_{VCG}^{\theta_i,\alpha_k,m_i}$ should be described as below:

$$Pay_{VCG}^{\theta_i,\alpha_k,m_i} = \sum_{\Theta\setminus\theta_i}^{\mathbb{K}} (b_j \cdot \beta_j \sum_{k=j_0}^{k+m_j-1} \alpha_k) - \sum_{\Theta\setminus\theta_i}^{\mathbb{K}\setminus\sum_{\alpha_k}^{\alpha_{k+m_i-1}}} (b_j \cdot \beta_j \sum_{k=j_0}^{k+m_j-1} \alpha_k) \tag{4}$$

According to the definition of payment, $Pay_{VCG}^{\theta_i,\alpha_k,m_i} = p_{VCG}^{\theta_i,\alpha_k,m_i} \cdot \beta_i \cdot \sum_k^{k+m_i-1} \alpha_k$, therefore, each time a user clicks on θ_i's link, the bidder has to pay

$$p_{VCG}^{\theta_i,\alpha_k,m_i} = \frac{Pay_{VCG}^{\theta_i,\alpha_k,m_i}}{\beta_i \cdot \sum_k^{k+m_i-1} \alpha_k} \tag{5}$$

3.3 Mixed GSP Mechanism

GSP is the dominant ads auction mechanism in sponsored search market much because of the ease of understanding its rules. A natural way to design a GSP-like mechanism in multi-slot auction is similar to the traditional GSP. We sort the ads according to their score, which is the product of β_i and b_i, and allocate all the slots according to this order.

Allocation Rule. In this mechanism, the K advertising slots are allocated to bidders in descending orders of their scores s_i.We follow the OPT allocation rule, which is a greedy algorithm and guarantees the social welfare maximum. From the top slot in $\mathbb{K}$, each step we choose a bidder with the maximum s_i to fill the slot. The allocation rule is the same as Algorithm 1.

Pricing Rule. In this mechanism we designed, each time a user clicks on a sponsored link, the corresponding bidder's account is automatically billed certain amount of money, according to her β_i and s_{i+1} of the advertiser who is just below her in the ordered queue. If the allocation rule is followed strictly, the price a bidder pays will be never more than what she bids. The pricing rule can be described as below:

$$p_{MixedGSP}^{\theta_i,\alpha_k,m_i} = \frac{s_{i+1}}{\beta_i} = \frac{\beta_{i+1}}{\beta_i} b_{i+1} \tag{6}$$

Where b_{i+1} denotes the bidding price of the bidder who is just below θ_i. The payment rule is exactly the same as the standard weighted GSP payment: $p_i = \frac{\beta_{i+1}}{\beta_i} b_{i+1}$.

Theorem 2. *Forward Nash equilibrium exists in Mixed GSP mechanism, where the adjusted bids are:*

$$\mathcal{F}_{\theta_i}(\Theta \backslash \theta_i) = \begin{cases} v_i - \frac{\sum_{j=k}^{k+m_i-1} \alpha_j}{\sum_{j=k-m_{i-1}}^{k-m_{i-1}+m_{i-1}} \alpha_j} [v_i - \frac{\beta_{i+1}}{\beta_i} b_{i+1}] & 2 \le i \le K \\ v_i & i = 1 \text{ or } i > K \end{cases} \tag{7}$$

Proof. When $\theta_i(v_i, b_i, \beta_i, m_i)$ get slot(s) $\{\alpha_k, \cdots, \alpha_{k+m-1}\}$, the utility is defined as:

$$u_{\alpha_k}^{\theta_i} = [v_i - p(b_i, a_k, m_i)] \times \beta_i \sum_{j=k}^{k+m_i-1} \alpha_j = [v_i - \frac{\beta_{i+1}}{\beta_i} b_{i+1}] \times \beta_i \sum_{j=k}^{k+m_i-1} \alpha_j \tag{8}$$

When the bidder gets slot(s) $\{\alpha_t, \cdots, \alpha_{t+m_i-1}\}$, since $p(b_i, \alpha_t, m_i) \leq b_i$, her utility in the worst case is:

$$u_{\alpha_t}^{\theta_i} = [v_i - p(b_i, a_t, m_i)] \times \beta_i \sum_{j=t}^{t+m_i-1} \alpha_j = [v_i - b_i] \times \beta_i \sum_{j=t}^{t+m_i-1} \alpha_j$$

$$\forall b^{-i},\ \forall t < k,\ u_{a_k}^{\theta_i(b_i,\beta_i,m_i)} \leq u_{a_t}^{\theta_i(b_i,\beta_i,m_i)}$$

$$\Leftrightarrow [v_i - \frac{\beta_{i+1}}{\beta_i} b_{i+1}] \times \beta_i \sum_{j=k}^{k+m_i-1} \alpha_j \leq [v_i - b_i] \times \beta_i \sum_{j=t}^{t+m_i-1} \alpha_j$$

$$\Leftrightarrow b_i \leq v_i - \frac{\sum_{j=k}^{k+m_i-1} \alpha_j}{\sum_{j=t}^{t+m_i-1} \alpha_j} [v_i - \frac{\beta_{i+1}}{\beta_i} b_{i+1}]$$

Suppose $\theta_i(v_i, b_i, \beta_i, m_i)$ is just below $\theta_{i-1}(v_{i-1}, b_{i-1}, \beta_{i-1}, m_{i-1})$, and since $\forall t \leq k - m_{i-1}, \sum_{j=t}^{t+m_i-1} \alpha_j \geq \sum_{j=k-m_{i-1}}^{k-m_{i-1}+m_i-1} \alpha_j$

$$\Leftrightarrow b_i \leq v_i - \frac{\sum_{j=k}^{k+m_i-1} \alpha_j}{\sum_{j=k-m_{i-1}}^{k-m_{i-1}+m_i-1} \alpha_j} [v_i - \frac{\beta_{i+1}}{\beta_i} b_{i+1}]$$

If the bidder preferred the highest slot, she should bid as high as possible in this range so that she could still obtain the highest slot after other bidders' responses. Similarly, as for the losers, she should bid her true value in order to get some slots after others' responses. Suppose $\mathcal{O}_{\theta_i}(\mathcal{M}_{\theta_i}(\Theta \backslash \theta_i), \Theta \backslash \theta_i) = k$, then the Forward-looking best response function is defined as equation(7).

3.4 Hybrid GSP Mechanism

The weakness of mixed GSP is that it does not follow the positive correlation of the bidder's bidding price and CTR they get, so it does not mix well. Therefore, we proposed a new GSP-like mechanism, that is, Hybrid GSP Mechanism.

Allocation Rule. In this mechanism, the K advertising slots are allocated to bidders, according to the descending order of the ratio of bidder's score s_i and the position factor α of its occupied slots, which we denote as unit bid ρ_{θ_i,k,m_i}

$$\rho_{\theta_i,k,m_i} = \frac{\beta_i \cdot b_i}{\sum_{j=k}^{k+m_i-1} \alpha_j} \tag{9}$$

Especially, for a single-slot bidder θ_{t_i} bidding for the k-th slot, $\rho_{\theta_{t_i},k,1} = \frac{\beta_{t_i} b_{t_i}}{\alpha_k}$, for another multi-slot bidder θ_{m_i} bidding for the continuous 3 slots following the k-th slot, $\rho_{\theta_{m_i},k,3} = \frac{\beta_{m_i} b_{m_i}}{\alpha_k + \alpha_{k+1} + \alpha_{k+2}}$.

We follow an allocation rule that from the top slot in $\mathbb{K}$, each time we choose an ad with the maximum ρ_{θ_i,k,m_i} to fill the slots as described in Algorithm 2.

Algorithm 2. Hybrid GSP Allocation Rule

Data: $\mathbb{K}\{\alpha_1, \alpha_2, \alpha_3 \cdots, \alpha_k\}$, $\mathrm{T}\{\theta_{t_1}, \theta_{t_2}, \cdots, \theta_{t_q}\}$, $\mathrm{M}\{\theta_{m_1}, \theta_{m_2}, \cdots \theta_{m_r}\}$
Result: An ordered list$< \theta >\Re$ of allocation for K slots
begin
  for $i \leftarrow 1$ **to** k **do**
    $\rho_t \leftarrow Max(\frac{b_j \cdot \beta_j}{\alpha_i} \mid \theta_j \in T), \rho_m \leftarrow Max(\frac{b_j \cdot \beta_j}{\alpha_i + \alpha_{i+1} + \alpha_{i+2}} \mid \theta_j \in M)$
    if *i>=k-2 or (i <k-2 and* $\rho_t > \rho_m$*)* **then**
      Insert θ_t to List $\Re_0$
      Remove θ_t from T
    else
      Insert θ_m to List $\Re_0$
      Remove θ_m from M
      $i \leftarrow i + 2$
  return $\Re_0$
end

Pricing Rule. In this mechanism we designed, each time a user clicks on a sponsored link, the corresponding bidder's account is automatically billed certain amount of money, according to the product of position factors α of those slots it occupies and $\rho_{\theta_{i+1},k,m_{i+1}}$ of the advertiser who is just below her in the ordered queue. If the allocation rule is executed strictly, the price a bidder pays will never be more than what she bids. The pricing rule can be described as below:

$$p_{HybridGSP}^{\theta_i,\alpha_k,m_i} = \frac{\rho_{\theta_{i+1},k,m_{i+1}}}{\beta_i} \times \sum_{j=k}^{k+m_i-1} \alpha_j = \frac{\beta_{i+1} \cdot \sum_{j=k}^{k+m_i-1} \alpha_j}{\beta_i \cdot \sum_{j=k}^{k+m_{i+1}-1} \alpha_j} b_{i+1} \tag{10}$$

Where b_{i+1} denotes the bidding price of the bidder who is just below θ_i in the ordered queue, m_{i+1} denotes the number of slots that θ_{i+1} needs.

Theorem 3. *Forward Nash equilibrium exists in Hybrid GSP mechanism, where the adjusted bids are:*

$$\mathcal{F}_{\theta_i}(\Theta \backslash \theta_i) = \begin{cases} v_i - \frac{\sum_{j=k}^{k+m_{i-1}} \alpha_j}{\sum_{j=k-m_{i-1}}^{k-m_{i-1}+m_i-1} \alpha_j} [v_i - \frac{\rho_{\theta_{i+1},k,m_{i+1}}}{\beta_i} \times \sum_{j=k}^{k+m_i-1} \alpha_i] & 2 \le i \le K \\ v_i & i=1 \text{ or } i > K \end{cases} \tag{11}$$

Proof. When $\theta_i(v_i, b_i, \beta_i, m_i)$ get slot(s) $\{\alpha_k, \cdots, \alpha_{k+m_i-1}\}$, the utility is defined as:

$$\begin{aligned} u_{\alpha_k}^{\theta_i} &= [v_i - p(b_i, \alpha_k, m_i)] \times \beta_i \sum_{j=k}^{k+m_i-1} \alpha_j \\ &= [v_i - \frac{\rho_{\theta_{i+1},k,m_{i+1}}}{\beta_i} \times \sum_{j=k}^{k+m_i-1} \alpha_i] \times \beta_i \sum_{j=k}^{k+m_i-1} \alpha_j \end{aligned} \tag{12}$$

When the bidder gets slot(s) $\{\alpha_t, \cdots, \alpha_{t+m_i-1}\}$, since $p(b_i, \alpha_t, m_i) \leq b_i$ her utility in the worst case is:

$$u_{\alpha_t}^{\theta_i} = [v_i - p(b_i, a_t, m_i)] \times \beta_i \sum_{j=t}^{t+m_i-1} \alpha_j = [v_i - b_i] \times \beta_i \sum_{j=t}^{t+m_i-1} \alpha_j$$

$$\forall b^{-i},\ \forall t < k,\ u_{a_k}^{\theta_i(b_i,\beta_i,m_i)} \leq u_{a_t}^{\theta_i(b_i,\beta_i,m_i)}$$

$$\Leftrightarrow [v_i - \frac{\rho_{\theta_{i+1},k,m_{i+1}}}{\beta_i} \times \sum_{j=k}^{k+m_i-1} \alpha_i] \times \beta_i \sum_{j=k}^{k+m_i-1} \alpha_j \leq [v_i - b_i] \times \beta_i \sum_{j=t}^{t+m_i-1} \alpha_j$$

$$\Leftrightarrow b_i \leq v_i - \frac{\sum_{j=k}^{k+m_i-1} \alpha_j}{\sum_{j=t}^{t+m_i-1} \alpha_j} [v_i - \frac{\rho_{\theta_{i+1},k,m_{i+1}}}{\beta_i} \times \sum_{j=k}^{k+m_i-1} \alpha_i]$$

Suppose $\theta_i(v_i, b_i, \beta_i, m_i)$ is just below $\theta_{i-1}(v_{i-1}, b_{i-1}, \beta_{i-1}, m_{i-1})$, and since $\forall t \leq k - m_{i-1}, \sum_{j=t}^{t+m_i-1} \alpha_j \geq \sum_{j=k-m_{i-1}}^{k-m_{i-1}+m_i-1} \alpha_j$

$$\Leftrightarrow b_i \leq v_i - \frac{\sum_{j=k}^{k+m_i-1} \alpha_j}{\sum_{j=k-m_{i-1}}^{k-m_{i-1}+m_i-1} \alpha_j} [v_i - \frac{\rho_{\theta_{i+1},k,m_{i+1}}}{\beta_i} \times \sum_{j=k}^{k+m_i-1} \alpha_i]$$

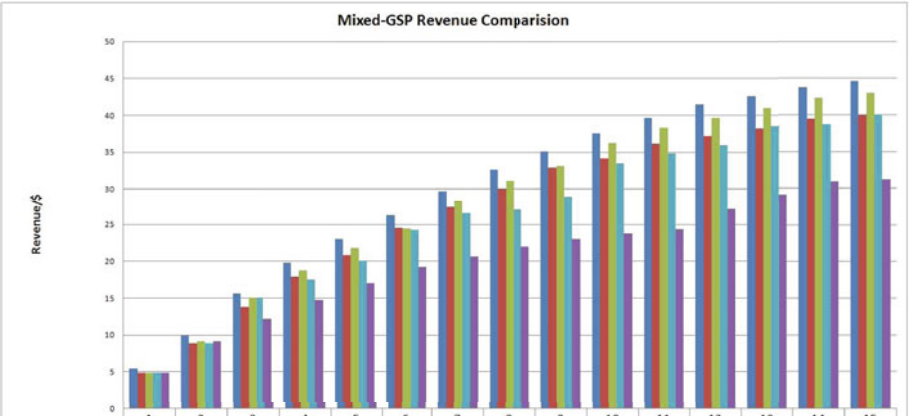
If the bidder preferred the highest slot, she should bid as high as possible in this range so that she could still obtain the highest slot after other bidders' responses. Similarly, as for the losers, she should bid her true value in order to get some slots after others' responses. Suppose $\mathcal{O}_{\theta_i}(\mathcal{M}_{\theta_i}(\Theta \backslash \theta_i), \Theta \backslash \theta_i) = k$, then the Forward-looking best response function is defined as equation(11).

4 Simulations for Revenue Comparison

In order to compare revenues of the mixed ads auction and the equivalent pure single-slot ads auction, we should find a fair way to convert all multi-slot ads to single-slot ads. A reasonable way to do it is to keep their ranks the same in both auctions.

We did a simulation to compare revenues under different mechanisms. The ads data we used was from two samples of a search engine, including 202 entries of single-slot ads and 24 entries of multi-slot ads, and we built the α-list from statistical data. For fair and more universal significance, we set all the ads' β values equal to 1, which means that the all the GSP-like mechanisms are non-weighted. The results of expected revenues under different mechanisms or equilibrium price are as below:

a) Optimal: optimal mechanism
b) VCG: VCG mechanism
   For Fig.2:
c) Mixed GSP: mixed GSP mechanism



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Optimal	5.424	9.896	15.65	19.778	23.046	26.27	29.678	32.602	35.062	37.554	39.661	41.424	42.594	43.735	44.596
VCG	4.859	8.915	13.832	17.864	20.828	24.596	27.356	30.008	32.898	34.177	36.145	37.224	38.257	39.489	39.906
Mixed-GSP	4.859	9.123	15.024	18.722	21.838	24.469	28.354	31.142	33.062	36.271	38.28	39.656	41.012	42.306	42.978
Forward-Looking	4.859	8.915	15.024	17.5309	20.0739	24.233	26.5578	27.0349	28.8884	33.5199	34.8292	35.9679	38.5631	38.7405	40.0377
Equivalent	4.859	9.123	12.195	14.775	17.055	19.275	20.675	22.035	22.995	23.76	24.397	27.145	29.123	30.932	31.184

Fig. 2. Simulation Results: Mixed GSP with other Mechanisms

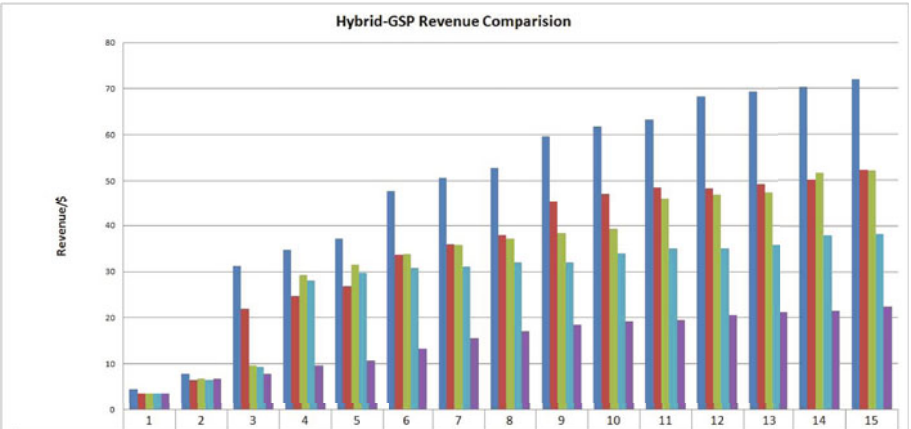
d) Forward-Looking: forward-looking Nash equilibrium price under mixed GSP mechanism. In this simulation, we calculated the bidders forward-looking Nash equilibrium price from bottom to up in the rank
e) Equivalent: the single-slot GSP mechanism, which the bidders' rank equivalent to the mixed GSP ranking. The converted multi-slot ad's bidding price is what they bid for multi-slot ads.

For Fig.3:

f) Hybrid GSP: hybrid GSP mechanism
g) Forward-Looking: forward-looking Nash equilibrium price under hybrid GSP mechanism. In this simulation, we caculated the bidders forward-looking Nash equilibrium price from bottom to up in the rank
h) Equivalent: the single-slot GSP mechanism, which the bidders' rank equivalent to the hybrid GSP ranking.The converted multi-slot ad's bidding price is $b_{i-1} - \epsilon$, where b_{i-1} is the bidding price of the nearest single-slot bid above it.

For each mechanism, we compared their revenues given different number of available slots, from 1 to 15.

From the simulation results, we found that the revenue under Mixed GSP and Hybrid GSP mechanism, are both greater than the revenue of equivalent single-slot GSP, when the slots quota is greater than 2. The performance of Hybrid GSP is better since its difference with equivalent single-slot GSP is greater, which indicates that Hybrid GSP may make more profits with the same slots.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Optimal	4.52	7.848	31.3	34.74	37.172	47.82	50.62	52.796	59.7	61.74	63.308	68.28	69.4	70.232	72.03
VCG	3.616	6.528	21.91	24.662	26.79	33.71	35.95	37.854	45.59	47.222	48.594	48.45	49.346	50.074	52.2
Hybrid GSP	3.616	6.736	9.616	29.2476	31.5276	33.7476	35.8476	37.2076	38.4076	39.2236	46.1892	47.0492	47.4972	51.6321	52.0521
Forward-Looking	3.616	6.528	9.408	28.0085	29.7196	30.8145	31.047	31.9396	32.0145	34.0396	34.975	34.9938	35.815	37.8651	38.2851
Equivalent	3.616	6.736	7.84687	9.616	10.7269	13.3069	15.5869	17.0669	18.4669	19.2061	19.5549	20.5661	21.2647	21.5261	22.4647

Fig. 3. Simulation Results: Hybrid GSP with other Mechanisms

5 Conclusions and Future Work

In this paper, we consider about a rich ads auction problem in sponsored search markets. We first abstracted this problem to a multi-slot ads auction model, solved it under Optimal mechanism and VCG mechanism, then we proposed two GSP-like mechanisms, denoted as Mixed GSP and Hybrid GSP respectively, and proved that forward-looking Nash equilibrium exists in both mechanisms.

Besides, we did a simulation for full comparison of expected revenues among payment rules including Optimal, VCG, Mixed GSP, its equivalent single-slot GSP and forward-looking Nash equilibrium price, Hybrid GSP, its equivalent single-slot GSP and also its forward-looking Nash equilibrium price.

Most importantly, via simulation, both the Mixed GSP mechanism and Hybrid GSP mechanism were shown to make more profits than their equivalent pure single-slot GSP mechanism, under the condition that all bidders get the same rank in all mechanisms.

Future Work. The real world rich ads auction in sponsored search market is far more complex than the model we built in this paper. For our primary model, we propose some issues and problems, and discussed below:

- In the present formulation of the rich ads auction problem, we have focused on the ads in single-slot or fixed number of slots, namely, $m_i \in \{1, n\}, n \in \mathbb{N}^+$, which is a common method used by search engines. However, under practice, different rich ads may have different sizes, that is, $m_i \in \mathbb{N}^+$. In this condition, does a polynomial-time algorithm exist, to allocate the ads and keep the social welfare maximal?

- In practice, each advertiser usually set a budget. The auctioneer will not allocate any slot to advertisers whose budgets are exhausted. The budget constraint gives a new perspective to the rich ads problem.

References

1. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. Am. Econ. Rev. 97(1), 242–259 (2007)
2. Varian, H.R.: Position auctions. Int. J. Ind. Organ. 25(6), 1163–1178 (2007)
3. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press, New York (2007)
4. Narahari, Y., Garg, D., Rama Suri, N., Prakash, H.: Game Theoretic Problems in Network Economics and Mechanism Design. Springer Press, Heidelberg (2009)
5. Bu, T., Liang, L., Qi, Q.: On Robustness of Forward-looking in Sponsored Search Auction. Algorithmica (2009)
6. Bu, T., Deng, X., Qi, Q.: Dynamics of strategic manipulation in ad-words auction. In: 3rd Workshop on Sponsored Search Auctions (SSA), in conjunction with WWW 2007, Banff, Canada (May 8, 2007)
7. Bu, T., Deng, X., Qi, Q.: Forward Looking Nash Equilibrium for Keyword Auction. Information Processing Letters 105, 41–46 (2008)
8. Vickrey, W.S.: Counterspeculation, auctions, and competitive sealed tenders. Journal of Finance XVI, 8–37 (1961)
9. Chen, X., Deng, X., Liu, B.J.: On incentive compatible competitive selection protocol. In: Proceedings of the 12th Annual International Computing and Combinatorics Conference (COCOON 2006), Taipei, Taiwan, August 15-18, pp. 13–22 (2006)
10. Zhou, Y., Rajan, L.: Vindictive bidding in keyword auctions. In: 2nd Workshop on Sponsored Search Auctions (SSA), in Conjunction with the ACM Conference on Electronic Commerce (EC 2006), Ann Arbor, USA (June 11, 2006)
11. Yahoo! Search Marketing blog, http://www.ysmblog.com/blog/2009/02/18/your-ads-richer/

Truthful Auction for CPU Time Slots*

Qiang Zhang and Minming Li

Department of Computer Science, City University of Hong Kong,
Tat Chee Avenue, Kowloon, Hong Kong
qianzhang8@student.cityu.edu.hk, minmli@cs.cityu.edu.hk

Abstract. We consider the task of designing a truthful auction mechanism for CPU time scheduling problem. There are m commodities (time slots) $T = \{t_1, t_2, \ldots, t_m\}$ for n buyers $I = \{1, 2, \ldots, n\}$. Each buyer requires a number of time slots s_i for its task. The valuation function of buyer i for a bundle of time slots T_i is $v_i(T_i) = w_i(m - t)$, where t is the last time slot in T_i and $|T_i| = s_i$. The utility u_i of buyer i is $v_i(T_i) - p(T_i)$. It is well-known that Vickrey-Clarke-Groves (VCG) mechanism gives the incentive to bid truthfully. Although optimal social welfare is computationally feasible in CPU time scheduling problem, VCG mechanism may produce low revenue. We design an auction which also maintains the incentives for bidders to bid truthfully. In addition, we perform simulations and observe that our truthful mechanism produces more revenue than VCG on average.

Keywords: Mechanism Design, Truthful Mechanism, Auctions, Scheduling.

1 Introduction

Internet market has become an important part of today's economy, which produces large amounts of revenue. Information or digital goods in the Internet market differ from the traditional market in some key aspects, for example, digital goods can be re-created at a marginal cost and the supply is unlimited. The pricing mechanism for digital goods also differs from that of classic economics since supply exceeds demand in the Internet market while supply is sold out in classic economics. This new economy system demands many theoretical analyses from which we can develop tools that help us determine the pricing mechanism and resource allocation. [5] gives an offline competitive auction for selling multiple digital goods where each bidder has a different private value for each digital good and bidders are willing to submit their true private values as bids. In mechanism design, such auction is called *truthful auction*, or *incentive-compatible auction*. [1] extends this concept to an online auction in which bidders are willing to bid their true valuations for each digital good.

In this work, we study the auction for CPU processing time as a digital good in the Internet market. CPU time is a crucial resource and has been extensively

* This work was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 117408].

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 23–34, 2010.

studied in grid computing. Market-oriented methods have been proposed for efficient allocation of computational resources in [2,9,7]. In most of CPU time allocation models, CPU time is considered as the same goods that can reduce the computational complexity. However, in practice bidders in the same market might have different valuations for different time slots. Furthermore, we assume all bidders are task-oriented, which means the valuations of time slot allocations for bidders depend on the finish time of their jobs. We adopt the model from [4] which considers each CPU time slot as a different unit. [4] focuses on the Walrasian equilibrium price and the complexity of computing such price. In this work, we are interested in incentive-compatible mechanisms for selling CPU time in the Internet market. Truth-telling is a desirable feature which ensures behavior is predictable in auction. Information about others' bids is irrelevant in a truthful auction so that bidders will not be motivated to spend resource on obtaining such information. It is well known that Vickrey-Clarke-Groves (VCG) mechanism [3,6,8] is a truthful auction mechanism which maximizes the social welfare. Nevertheless, VCG mechanism has many notable weaknesses, for example, VCG is computationally infeasible in many problems and the payment of VCG mechanism could be very low. We first develop a truthful auction for time slots on single CPU, in which the CPU time slots allocation mechanism is a greedy function that depends on the valuation from each bidder. We also discuss the revenue from our auction mechanism and compare it with the revenue of VCG. Finally, we extend the mechanism to selling processing time on multiple CPUs with an assumption that bidders are only allowed to submit their valuation parameters untruthfully. We proved that the proposed auction model is also *incentive-compatible* in multiple CPU setting. The paper is organized as follows. The CPU time scheduling problem is described in Section 2. In Section 3, we propose a new auction for selling CPU time on a single CPU. We describe the pricing and allocation mechanism. In Section 4, we prove that the auction is *incentive-compatible*. In Section 5, we analyze the revenue generated from our auction mechanism. We also perform simulations and compare the revenue between our mechanism and VCG. In Section 6, we extend our mechanism and develop an *incentive-compatible* auction for multiple CPUs. At last, we conclude our work with discussion on the results and possible future work in Section 7.

2 The CPU Time Scheduling Problem

In the CPU time scheduling problem, there is a CPU time provider and many bidders in the market. CPU time provider sells CPU time slots to bidders and each bidder would like to buy different number of CPU time slots that depends on the requirement of its job. The valuation of each bidder depends on the time slots it obtains. The CPU time scheduling problem can be formulated as follows. Assume there are n jobs announced at the time $t = 1$, the j^{th} job needs time $s_j > 0$ to complete its task and has weight $w_j > 0$. From these environment settings, We form an exchange economy: an auctioneer sells m commodities (time slots) $T = \{t_1, t_2, \ldots, t_m\}$ to n buyers (jobs) $I = \{1, 2, \ldots, n\}$. The valuation

function of buyer i for a bundle of time slots T_i is $v_i(T_i) = w_i(m - t)$, where t is the largest time-slot item in T_i and $|T_i| = s_i$. The utility u_i of buyer i is $v_i(T_i) - p(T_i)$.

3 A Truthful Auction for Time Slots on Single CPU

In this section, we present a truthful auction mechanism for pricing time slots on single CPU with a linear valuation function for each job. As mentioned in the last section, the valuation function of buyer i for a bundle of time slots T_i is $v_i(T_i) = w_i(m-t)$, where t is the largest time-slot item in T_i and $|T_i| = s_i$. The utility u_i of buyer i is $v_i(T_i) - p(T_i)$. The goal of the truthful auction is to give each buyer an incentive to submit its true value of w and s.

The truthful auction we propose is as follows:

Rank buyers according to their submitted weight w. Without loss of generality, assume that $w_1 \geq w_2 \ldots \geq w_n$. Assign time slots to buyers according to their ranks, i.e. buyer 1 takes the first s_1 time slots, followed by s_2 time slots assigned to buyer 2 and so on. We denote the time slots allocation as A, $K_j(A) = i$ if and only if time slot t_j is assigned to buyer i, and $B_j(A) = \{j+1, j+2, \ldots, m\}$

If the bundle of time slots assigned to buyer i is

$$T_i(A) = \{t_\gamma, t_{\gamma+1}, \ldots, t_{\eta=\gamma+s_i-1}\} \tag{1}$$

where γ_i and η_i are the start and finish time of buyer i, respectively. In this case, the price of the bundle of time slots is

$$p(T_i(A)) = \sum_{j \in B_{\eta_i}(A)} w_{K_j(A)} \tag{2}$$

As an example, consider four buyers bidding for 6 time slots $t_1, t_2, \ldots, t_6$. The time span and weight are $\{2, 3, 1\}$ and $\{9, 6, 4\}$. The time slots s_1 and s_2 will be assigned to buyer 1. By Equation 2, the price of the bundle of time slots t_1 and t_2 is $p(T_1) = w_2 s_2 + w_3 s_3 = 6 * 3 + 4 * 1 = 22$. Applying the same method to buyer 2, the price of the bundle of time slots t_3, t_4, t_5 is $p(T_2) = w_3 s_3 = 4$. Because there are no more buyers, buyer 3 is charged at price 0 for its time slots t_6.

4 Truthfulness of the Auction

In this section, we prove the pricing and allocation mechanism proposed in Section 3 is *incentive-compatible*. From the proof, we show that there is no other bidding strategies rather than bidding truthfully which could gain more utility.

Definition 1. *A mechanism $(f, p_1, \ldots, p_n)$ is called incentive compatible or truthful if for every buyer i, every $w_1 \in W_1, \ldots, w_n \in W_n$, every $s_1 \in S_1, \ldots, s_n \in S_n$, and every $w_i^{'} \in W_i, s_i^{'} \in S_i$, if we denote $a = f(w_i, s_i, w_{-i}, s_{-i})$ and $a^{'} = f(w_i^{'}, s_i^{'}, w_{-i}, s_{-i})$, then $v_i(a) - p_i(w_i, s_i, w_{-i}, s_{-i}) \geq v_i(a^{'}) - p_i(w_i^{'}, s_i^{'}, w_{-i}, s_{-i})$, where W_i, S_i is the set of all possible values of w_i, s_i for buyer i, and w_{-i}, s_{-i} is the bid vector except buyer i.*

Algorithm 1. Auction mechanism for time slots on single CPU

Input: A set of jobs $I = \{1, 2, \ldots, n\}$ with weight w and timespan s ordered by w

Output: A set of prices for each job

```
begin
    count_timeslot = 1 ;
    for i = 1 to n do
        start_time_i = count_timeslot;
        finish_time_i = count_timeslot + s_i − 1;
        count_timeslot = count_timeslot + s_i;
        for j ← i + 1 to n do
            p(T_i) = p(T_i) + w_j s_j
end
```

Theorem 1. *The auction mechanism in Algorithm* 1 *is incentive-compatible.*

Proof. Fix i, the valuation function $v_i = w_i(m - t)$, w_{-i} and s_{-i}, we need to show that the utility when buyer i declares w_i and s_i is not less than the utility when he declares a fake $w_i^{'}$ and $s_i^{'}$.

If buyer i submits a $w_i^{'} > w_i$, it will possibly improve the ranking of buyer i. Suppose the original ranking of buyer i when declaring w_i is k, then the new ranking is $k^{'} < k$ by submitting $w_i{'}$. Let $Q = \{\rho, \ldots, \sigma\}$ where the rankings of users $\rho, \ldots, \sigma$ are from $k^{'} + 1$ to k when buyer i submits $w_i^{'}$. Therefore, for buyer i, its completion time improves by $\sum_{j \in Q} s_j$ but it need to pay $\sum_{j \in Q} w_j s_j$ more. Its utility improves by:

$$\begin{aligned} & v_i(a^{'}) - p_i(w_i^{'}, s_i^{'}, w_{-i}, s_{-i}) - (v_i(a) - p_i(w_i, s_i, w_{-i}, s_{-i})) \\ =& (v_i(a^{'}) - v_i(a)) - (p_i(w_i^{'}, s_i^{'}, w_{-i}, s_{-i}) - p_i(w_i, s_i, w_{-i}, s_{-i})) \\ =& w_i \sum_{j \in Q} s_j - \sum_{j \in Q} w_j s_j \end{aligned} \tag{3}$$

Because all buyers in Q has an equal or higher w than buyer i, Equation 3 will produce a non-positive result, which means the utility of buyer i will not increase if it submits a $w_i^{'} > w_i$.

In the same way, if buyer i submits a $w_i^{'} < w_i$, then the new ranking is $k^{''} > k$. Let $Q^{'} = \{\rho^{'}, \ldots, \sigma^{'}\}$ where the rankings of users $\rho^{'}, \ldots, \sigma^{'}$ are from k to $k^{''} - 1$ when buyer i submits $w_i^{'}$. Therefore, for buyer i, its completion time delays $\sum_{j \in Q'} s_j$ but it can pay $\sum_{j \in Q'} w_j s_j$ less. Its utility improves by:

$$\begin{aligned} & v_i(a^{'}) - p_i(w_i^{'}, s_i^{'}, w_{-i}, s_{-i}) - (v_i(a) - p_i(w_i, s_i, w_{-i}, s_{-i})) \\ =& (v_i(a^{'}) - v_i(a)) - (p_i(w_i^{'}, s_i^{'}, w_{-i}, s_{-i}) - p_i(w_i, s_i, w_{-i}, s_{-i})) \\ =& - w_i \sum_{j \in Q'} s_j + \sum_{j \in Q'} w_j s_j \end{aligned} \tag{4}$$

Because all buyers in $Q^{'}$ has a less or equal value of w than buyer i, Equation 4 will produce a non positive result, which means the utility of buyer i will not increase if it submits a $w_i^{'} < w_i$.

It is noted that buyer i cannot improve its utility by submitting a fake $s_i^{'}$ whatever buyer i submits a $w_i^{'} \geq w_i$ or $w_i^{'} \leq w_i$. For a fixed w_i, if buyer i submits $s_i^{'} < s_i$, it cannot complete its task because it only obtains $s_i^{'}$ slots. If buyer i submits $s_i^{'} > s_i$, buyer i gets more time slots but pays the same price. However, the utility u_i does not increase because buyer i cannot finish its task earlier.

Theorem 2. *Bidding job weight and time span truthfully always gains non-negative utility.*

Proof. Assume that if buyer i submits w_i, s_i, the valuation of buyer i is

$$v_i = w_i(m - \beta) \tag{5}$$

According to Equation 2, the price that buyer i pays is only dependent on the buyers having lower weights.

$$p(T_i) = \sum_{j \in B_\beta(A)} w_{K_j(A)} \tag{6}$$

Because $w_i \geq w_{K_j(A)}$, where $j \in B_\beta(A)$, the utility u_i of buyer i is:

$$u_i = v_i - p(T_i) = w_i(m - \beta) - \sum_{j \in B_\beta(A)} w_{K_j(A)} \geq 0 \tag{7}$$

5 Revenue Study

After designing a truthful mechanism for pricing time slots on single CPU, it is worth evaluating how much the revenue seller may earn by our mechanism. This section presents a comparison on revenue between our mechanism and VCG, a well-known truthful mechanism.

5.1 Experimental Results

From the formula presented in Section 3, we cannot tell which mechanism can achieve higher revenue. In order to further investigate the revenue by our truthful mechanism, we perform simulations at different number of bidders when the parameters (e.g. weight and time span) are normally distributed in a certain range. First, we choose both weight and time span parameters randomly from 1 to 100. We perform simulation 100 times at each number of bidders when the number of bidders increases from 2 to 100. In each simulation, we calculate

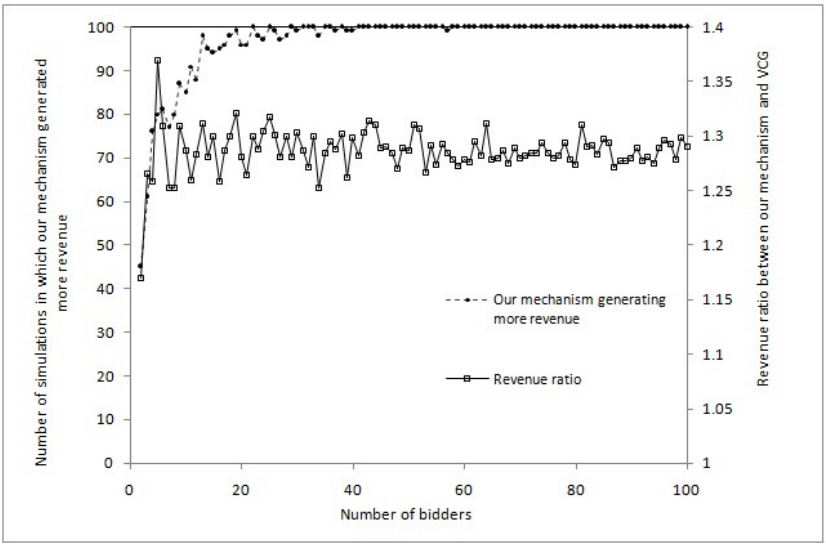


Fig. 1. Comparison of number of wins and revenue ratio between our truthful mechanism and VCG when both job weight and time span range from 1 to 100

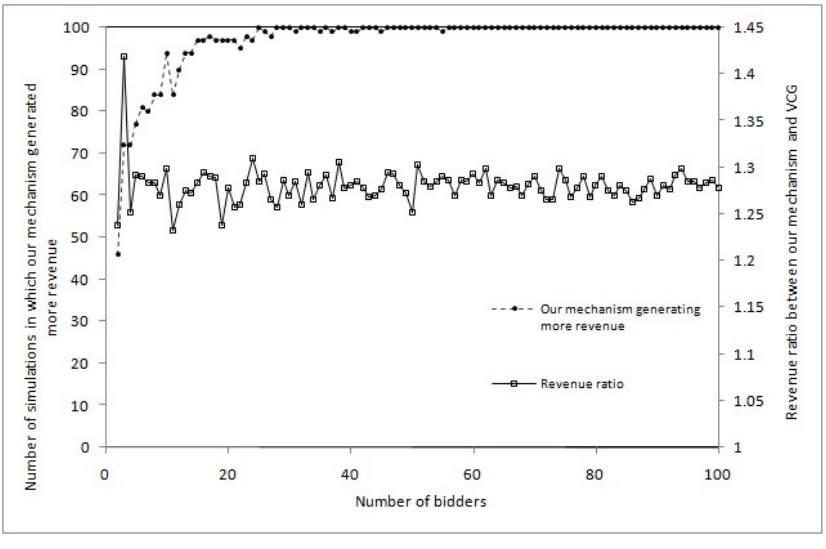


Fig. 2. Comparison of number of wins and revenue ratio between our truthful mechanism and VCG when job weight ranges from 1 to 100 and time span ranges from 1 to 50

the revenues from our mechanism and VCG and compare them. We count the number of times our mechanism generating more revenue than VCG. We also compute the average revenues of our mechanism and VCG when the number of bidders are fixed. Figure 1 shows the comparison result and the average revenue ratio on our mechanism and VCG. Second, we use the same method to compare our mechanism and VCG when weights are randomly chosen from 1 to 100 and time span ranges from 1 to 50. Figure 2 shows the comparison result and the average revenue ratio on our mechanism and VCG in this setting.

5.2 More Analyses

From the previous section, we find that our mechanism can generate more revenue than VCG when the parameters(e.g. w and s) are normally distributed. In this subsection we analyze the revenue generated on two special cases. One is that the valuation w_i of each buyer (job) grows with the time-span s_i it requires, the other is the valuations of buyers is inversely related to the time-span, i.e. the valuation w_i grows while the required time-span s_i decreases.

Case 1

$$\begin{cases} \text{a) } w_1 \leq w_2 \leq w_3 \leq \ldots \leq w_{n-1} \leq w_n \\ \text{b) } s_1 \leq s_2 \leq s_3 \leq \ldots \leq s_{n-1} \leq s_n \end{cases}$$

This scenario often happens when the valuation of job only depends on the timespan it needs. In practice, those jobs that need large amount of processing time most likely produce more value compared to other jobs. In this circumstance, our mechanism gives time slots according to the bidders' submitted weight w, e.g. bidder n gets the first s_n slots. Each bidder is charged by Equation 2, and the total revenue is:

$$(n-1)w_1s_1 + (n-2)w_2s_2 + \ldots + w_{n-1}s_{n-1} \tag{8}$$

While VCG mechanism allocates time slots in a socially optimal manner, e.g. bidder n with largest w/s obtains the first t_n slots. VCG mechanism charges each individual bidder the harm it causes to other bidders, e.g. bidder n is charged at price $s_n(w_1 + w_2 + \ldots + w_{n-1})$. There are different subcases in this scenario for the VCG mechanism. We study two special subcases.

The first subcase is $w_1/s_1 \leq w_2/s_2 \leq w_3/s_3 \leq \ldots \leq w_{n-1}/s_{n-1} \leq w_n/s_n$. In this subcase, the total revenue VCG mechanism generates is:

$$s_n(w_1 + w_2 + \ldots + w_{n-1}) + s_{n-1}(w_1 + w_2 + \ldots + w_{n-2}) + \ldots + s_2w_1 \tag{9}$$

We can show that the revenue from VCG mechanism is equal to or greater than our truthful mechanism. It can be proved from Equation 8 - Equation 9:

$$
\begin{aligned}
&(n-1)w_1s_1 + (n-2)w_2s_2 + \ldots + w_{n-1}s_{n-1} \\
&-(s_2+s_3+\ldots+s_n)w_1 - (s_3+\ldots+s_n)w_2 - \ldots - s_{n-1}w_n \\
=&((n-1)s_1 - s_2 - s_3 - \ldots - s_n)w_1 + ((n-2)s_2 - s_3 - \ldots - s_n)w_2 + \\
&\ldots + (s_{n-1} - s_n)w_{n-1} \\
\leq& 0
\end{aligned}
$$

Another subcase is that $w_1/s_1 \geq w_2/s_2 \geq w_3/s_3 \geq \ldots \geq w_{n-1}/s_{n-1} \geq w_n/s_n$. In this subcase, the total revenue VCG mechanism generates is:

$$
s_1(w_2+w_3+\ldots+w_n) + s_2(w_3+w_4+\ldots+w_n) + \ldots + s_{n-1}w_n \tag{10}
$$

The revenue comparison result is the same as subcase 1. It is because:

$$
\begin{aligned}
&(n-1)w_1s_1 + (n-2)w_2s_2 + \ldots + w_{n-1}s_{n-1} \\
&-(w_2+w_3+\ldots+w_n)s_1 - (w_3+w_4+\ldots+w_n)s_2 - \ldots - s_{n-1}w_n \\
=&((n-1)w_1 - w_2 - w_3 - \ldots - w_n)s_1 + ((n-2)w_2 - w_3 - \ldots - w_n)s_2 + \\
&\ldots + (w_{n-1} - w_n)s_{n-1} \\
\leq& 0
\end{aligned}
$$

Case 2

$$
\begin{cases}
\text{a) } w_1 \leq w_2 \leq w_3 \leq \ldots \leq w_{n-1} \leq w_n \\
\text{b) } s_1 \geq s_2 \geq s_3 \geq \ldots \geq s_{n-1} \geq s_n
\end{cases}
$$

This scenario often happens in the market when urgent jobs with high values and short timespans come into the market, while a number of jobs are being scheduled. For example, there are a lot of services that need a large amount of CPU time to process, but some customer-oriented services need to be attended promptly. This situation creates a problem of how to schedule this urgent job and how much it pays. In this case, we analyze the revenue when jobs in the market follow this pattern. Both our truthful mechanism and VCG mechanism produce the same time slots allocation and revenue formula as the first subcase in Case 1. However, the revenue from our truthful mechanism is equal to or greater than that from VCG mechanism. It is because time-span relationship differs from that in Case 1. The proof is as follows:

$$
\begin{aligned}
&(n-1)w_1s_1 + (n-2)w_2s_2 + \ldots + w_{n-1}s_{n-1} \\
&-(w_1+w_2+\ldots+w_{n-1})s_n - (w_1+w_2+\ldots+w_{n-2})s_{n-1} - \ldots - w_1s_2 \\
=&(n-1)w_1s_1 + (n-2)w_2s_2 + \ldots + w_{n-1}s_{n-1} \\
&-w_1(s_2+s_3+\ldots+s_{n-1}+s_n) - w_2(s_3+s_4+\ldots+s_{n-1}+s_n) - \ldots - w_{n-1}s_n \\
=&((n-1)s_1 - s_2 - s_3 - \ldots - s_n)w_1 + ((n-2)s_2 - s_3 - \ldots - s_n)w_2 + \\
&\ldots + (s_{n-1} - s_n)w_{n-1} \\
\geq& 0
\end{aligned}
$$

6 A Truthful Auction for Time Slots on Multiple CPUs

In this section, we extend our previous result to multiple CPUs setting. We propose a mechanism for pricing time slots on multiple CPUs, and prove that the mechanism is also *incentive compatible* when bidders are only allowed to submit a fake weight. The problem is formulated as follows: There are k CPUs, and an auctioneer sells m commodities (time slots) $T = \{t_1, t_2, \ldots, t_m\}$ on each CPU to n buyers (jobs) $I = \{1, 2, \ldots, n\}$. Therefore, the total number of time slots is km. Let T_{pq} be the p^{th} time slot on CPU q. Each buyer i needs time s_i to complete its task and has weight $w_i > 0$. The valuation function of buyer i for a bundle of time slots T_i is $v_i(T_i) = w_i(m - t)$, where t is the latest time-slot item in T_i. The utility u_i of buyer i is $v_i(T_i) - p(T_i)$. The goal of the mechanism is to give buyers an incentive to report its true value of w and s as a bid.

The solution to this problem is inherited from the idea on single CPU. The buyers are ordered by their submitted weights w. Without loss of generality, assume that $w_1 \geq w_2 \ldots \geq w_n$. Assign time slots on different CPUs to buyers according to the rank of their weights. We denote the time slot allocation as A and the bundle of time slots assigned to buyer i as $T_i(A) = \{t_{pq}, t_{p(q+1)}, ...t_{pk}, ...t_{p'k'}\}$ where $|T_i(A)| = s_i$, and p, $p^{'}$ are the start and finish time of buyer i. Let $B_i(A)$ be the time slots after the finish time of buyer i, i.e. $B_i(A) = \{p^{'} + 1, \ldots, m\}$. We also define $M_i(A)$ as the job index with maximum weight assigned to the i^{th} time slots among all CPUs. The price of the bundle of time slots $T_i(A)$ is

$$p(T_i(A)) = \sum_{j \in B_i(A)} w_{M_j(A)} \tag{11}$$

If the context is clear, we will omit A in equations. Suppose that there are 3 CPUs and 4 buyers, the time spans and weights are $\{2, 4, 5, 1\}$ and $\{9, 6, 4, 5\}$. In this example, buyer 1 is assigned to the first time slot of CPU 1 and 2; buyer 2 is assigned to the first time slot of CPU 3 and the second time slot of CPU 1, 2 and 3; buyer 3 is assigned to the third time slot of CPU 1, 2 and 3 and the fourth time slot of CPU 1 and 2; finally, buyer 4 is assigned to the fourth time slot of CPU 3. The intuition behind Equation 11 is that the price of the bundle of time slots for each buyer is dependent on the maximum weight buyer on every time slot among all CPUs after its finish time. In this example, buyer 1 pays $p(T_1) = \sum_{j \in B_i} w_{M_j} = w_2 + w_3 + w_3 = 14$. Applying the same method, the payments of buyer 2, 3 and 4 are 8, 0 and 0.

Theorem 3. *The auction mechanism in Algorithm* 2 *is incentive-compatible.*

Proof. Fix a buyer i, the valuation function $v_i = w_i(m - t)$, w_{-i} and s_{-i}, we show that the utility u_i when buyer i declares w_i is not less than the utility $u_i^{'}$ when buyer i declares a fake $w_i^{'}$. In this proof, we do not take buyers declaring a fake $s^{'}$ into consideration. It is because that buyer i cannot complete its task if $s_i^{'} < s_i$. If buyer i submits $s_i^{'} > s_i$, in practice, the auctioneer is able to check

whether each bidder submit its s truthfully or not from the process state or log file of the CPU. Therefore, the auctioneer can inflict penalty on buyer i in that case.

Algorithm 2. Auction mechanism for time-slot on multiple CPUs

Input: A set of jobs $I = \{1, 2, \ldots, n\}$ with weight w and timespan s ordered by w, A set of time slots $T = \{t_{11}, t_{12}, \ldots, t_{1k}, t_{21}, \ldots, t_{pk}\}$ on k CPUs
Output: A set of allocation T and a set of prices P

```
begin
    timeslot = 1 ;
    cpu = 1 ;
    for k ← 1 to n do
        for j ← 1 to s_k do
            T_k ← T_k ∪ t_{timeslot,cpu};
            finish_time_k = timeslot;
            if cpu = 1 then
                M_{timeslot} = k ;
            cpu ← cpu + 1;
            if cpu > k then
                cpu = 1 ;
                timeslot = timeslot + 1 ;
    for k ← 1 to n do
        for j ← finish_time_k + 1 to p do
            p(T_k) = p(T_k) + w_{M_j} ;
end
```

If buyer i bids $w_i^{'} > w_i$, it will increase the ranking of buyer i. Because of the multiple CPUs, the increase on the ranking might not advance the finish time of buyer i. It is because buyer i might finish its job on different CPUs at the same time as when it bids the true value. In this case, the price and valuation of buyer i remain unchanged. Otherwise, even buyer i will finish its job earlier, it will not improve the utility of buyer i. Suppose that the finish time is β when buyer i bids truthfully and $\beta^{'}$ if buyer submits $w_i^{'}$. The valuation of buyer i improves $w_i(\beta - \beta^{'})$. Nonetheless, buyer i need to pay $\sum_{j \in \{\beta^{'}+1,\ldots,\beta\}} w_{M_j}$ more. Hence,its utility improves by:

$$\begin{aligned} & v_i(T_i^{'}) - p_i(w_i^{'}, s_i^{'}, w_{-i}, s_{-i}) - (v_i(T_i) - p_i(w_i, s_i, w_{-i}, s_{-i})) \\ =& v_i(T_i^{'}) - v_i(T_i) - (p_i(w_i^{'}, s_i^{'}, w_{-i}, s_{-i}) - p_i(w_i, s_i, w_{-i}, s_{-i})) \\ =& w_i(\beta - \beta^{'}) - \sum_{j \in \{\beta^{'}+1,\ldots,\beta\}} w_{M_j} \end{aligned} \tag{12}$$

Because all buyers are scheduled by their submitted w, when buyer i reports a fake $w_i^{'} > w_i$, all jobs scheduled between $\beta^{'} + 1$ and β have a $w \geq w_i$. Therefore, Equation 12 always produces a non-positive result.

On the other hand, if buyer i reports $w_i^{'} < w_i$, it will lower the ranking of buyer i. The loss of ranking might not postpone the finish time of buyer i. In this case, the price and valuation of buyer i remain the same. If buyer i could accomplish its job at a time $\beta^{'} > \beta$ and pays less, it still will not improve the utility of buyer i. Reporting $w_i^{'}$ instead of w_i, the valuation of buyer i drops $w_i(\beta^{'} - \beta)$ and buyer i pays $\sum_{j \in \{\beta+1,\ldots,\beta'\}} w_{M_j}$ less. However, its utility improves by:

$$\begin{aligned} & v_i(T_i^{'}) - p_i(w_i^{'}, s_i^{'}, w_{-i}, s_{-i}) - (v_i(T_i) - p_i(w_i, s_i, w_{-i}, s_{-i})) \\ =& v_i(T_i^{'}) - v_i(T_i) - (p_i(w_i^{'}, s_i^{'}, w_{-i}, s_{-i}) - p_i(w_i, s_i, w_{-i}, s_{-i})) \\ =& - w_i(\beta^{'} - \beta) + \sum_{j \in \{\beta+1,\ldots,\beta'\}} w_{M_j} \end{aligned} \tag{13}$$

Because all buyers scheduled in time slots $\{\beta + 1, \ldots, \beta^{'}\}$ have weight $w \leq w_i$, Equation 13 will have a non-positive result, which means bidding a fake value $w^{'}$ will not improve buyer i's utility.

7 Conclusion and Discussion

In this paper, we present an incentive-compatible auction for selling processing time slots on single CPU. With the similar technique, we propose an auction for selling processing time slots on multiple CPUs where bidders are also willing to submit their true private values as bids. We believe our auction mechanism can generate more profit than the VCG mechanism on average. This study helps us to explore the possible truthful auction design when the CPU processing time slots is the commodity in the market. As cloud computing becomes a trend of future Internet computing, sharing or allocation of resource and computing power will be an important issue. The resource or computing power provider will not be exclusive in that market since each participant will play both roles of provider and buyer. The complexity of that market increases which leaves a question that how to design an incentive-compatible mechanism. We believe that it is a possible extension of our current work.

References

1. Bar-Yossef, Z., Hildrum, K., Wu, F.: Incentive-compatible online auctions for digital goods. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, p. 970 (2002)
2. Buyya, R., Stockinger, H., Giddy, J., Abramson, D.: Economic models for management of resources in grid computing. In: Technical Track on Commercial Applications for High-Performance Computing, SPIE International Symposium on The Convergence of Information Technologies and Communications, ITCom 2001 (2001)
3. Clarke, E.: Multipart pricing of public goods. Public choice 11(1), 17–33 (1971)
4. Deng, X., Huang, L., Li, M.: On walrasian price of cpu time. Algorithmica 48(2), 159–172 (2007)

5. Goldberg, A., Hartline, J., Wright, A.: Competitive auctions for multiple digital goods. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 416–427. Springer, Heidelberg (2001)
6. Groves, T.: Incentives in teams. Econometrica: Journal of the Econometric Society, 617–631 (1973)
7. Huberman, B., Hogg, T.: Distributed computation as an economic system. The Journal of Economic Perspectives, 141–152 (1995)
8. Vickrey, W.: Counterspeculation, auctions, and competitive sealed tenders. The Journal of Finance 16(1), 8–37 (1961)
9. Walsh, W., Wellman, M., Wurman, P., MacKie-Mason, J.: Some economics of market-based distributed scheduling. In: Eighteenth International Conference on Distributed Computing Systems (1998)

Top-d Rank Aggregation in Web Meta-search Engine
(Extended Abstract)

Qizhi Fang[1], Han Xiao[1], and Shanfeng Zhu[2]

[1] Department of Mathematics, Ocean University of China
Qingdao 266100, P.R. China
qfang@ouc.edu.cn, xiaohan@live.cn
[2] School of Computer Science and Shanghai Key Lab of Intelligent Information Processing, Fudan University, Shanghai 200433, P.R. China
zhushanfeng@gmail.com

Abstract. In this paper, we consider the rank aggregation problem for information retrieval over Web making use of a kind of metric, the coherence, which considers both the normalized Kendall-τ distance and the size of overlap between two partial rankings. In general, the top-d coherence aggregation problem is defined as: given collection of partial rankings $\Pi = \{\tau_1, \tau_2, \cdots, \tau_K\}$, how to find a final ranking π with specific length d, which maximizes the total coherence $\Phi(\pi, \Pi) = \sum_{i=1}^{K} \Phi(\pi, \tau_i)$. The corresponding complexity and algorithmic issues are discussed in this paper. Our main technical contribution is a polynomial time approximation scheme (PTAS) for a restricted top-d coherence aggregation problem.

Keywords: Rank aggregation, Kendall-τ distance, coherence, $\mathcal{NP}$-hard, approximate algorithm, PTAS.

1 Introduction

Meta-search engines are developed to overcome the shortcoming of single search engine and try to benefit from cooperate decision by combining the results of multiple independent search engines, that make use of different models and configurations. In this work, we focus on meta-search problem in which only ranking lists are provided by source search engines. This can be modeled by social choice theory that concentrates on how to aggregate individual's preferences into group's rational preferences. We can view the source search engines as voters, and all ranked documents as alternatives (candidates), then meta-search problem is actually to find a social choice function to obtain group's preferences on these documents (alternatives). Comparing with traditional voting problem, the rank aggregation problem on the web has some distinct features. Firstly, the number of voters is much less than the number of alternatives. Secondly, each search engine ranks a different set of web pages, determined by the different coverage of web search engines and various ranking algorithms adopted. A common

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 35–44, 2010.

case is that search engines only list the top d results, called top-d rankings, with respect to a query, and the other pages not listed can be assumed to be ranked below the top d results by the search engine.

The rank aggregation problem is to combine a profile of different rank orderings on a set of alternatives, in order to obtain a 'better' ranking. The notion of 'better' depends on what objective we strive to optimize. Dwork, et al.[5], made use of Kendall-τ distance as a criterion for meta-search on the web: Given a collection of partial rankings $\tau_1, \cdots, \tau_k$ of alternative web pages, they want to find a final ranking π of web pages, which minimize the sum of the Kendall-τ distance between π and τ_i $(i = 1, \cdots, k)$. The Kendall-τ distance between two ranking lists is the total number of pairs of alternatives that are assigned to different relative orders in the two ranking lists. In view of the shortage of the Kendall-τ distance for partial ranking lists with small size of overlap, Chin, et al.[4], proposed a new metric, coherence, considering both Kendall-τ distance and size of overlap of the partial ranking lists for an alternative measure for partial ranking aggregation. Under the metric of coherence, the goal of the rank aggregation is to maximize the sum of the coherence between π and τ_i $(i = 1, \cdots, k)$.

As described in [6,7,8,10] and other studies, people seldom go beyond top several pages of the result listed by a search engine, which means the alternative web pages at the top of any ranking are the most important to the users. Therefore, we concentrate our study on coherence aggregation problem of providing a partial ranking with the most important alternatives, top-d rankings (top-d CAP): For a given collection of partial rankings $\tau_1, \cdots, \tau_k$ (they may have different lengthes), we are interested in finding a final ranking π of specified length d such that the sum of coherence between the given rankings and the final ranking is maximum.

In this paper, we focus on the complexity and algorithmic issues for the top-d CAP. The main technical contribution is a polynomial time approximation scheme (PTAS) for top-d CAP under some reasonable restriction. Our approach is motivated by a unified framework of exhaustive sampling and transforming polynomial constrains into linear constrains for designing PTASs for polynomial integer programs with 'smooth' coeficients in [1,2]. In general, the coherence aggregation problems do not satisfy the 'smoothness' condition on the coefficients required in Arora's unified approach. Using an extended exhaustively sampling method, Chin, et. al.[4], obtained a PTAS for CAP with the final ranking being complete ranking. For the top-d CAP, however, the corresponding integer programming formulation has non-constant coefficients in the objective function. We further extend Arora's general methodology and provide a new insight into design of PTAS.

The structure of the paper is as follows. In section 2, we introduce the definitions and discuss the computational complexity of top-d CAP. Section 3 is dedicated to a simple heuristic algorithm with performance ratio 2. In Section 4, a polynomial time approximation scheme (PTAS) for top-d CAP is presented. In Section 5, we conclude with remarks and further discussions.

2 Definitions and Complexity

The formal definitions of coherence aggregation are as follows.

Given a set of alternatives $N = \{1, 2, \cdots, n\}$, a ranking π with respect to N is a permutation of some elements of N which represents a voter's or a judge's preference on these alternatives. For a ranking π, let N_π denote the set of elements presented in π, $|\pi| = |N_\pi|$ denote the number of elements in π, or the length of π. For each $i \in N_\pi$, $\pi(i)$ denote the position of the element i in π, and for any two elements $i, j \in N_\pi$, $\pi(i) < \pi(j)$ implies that i is ranked higher than j by π. If π orders all the elements in N, it is called a complete ranking; otherwise, a partial ranking.

Given two partial rankings π and τ, denote by $n[\pi, \tau] = |N_\pi \cap N_\tau|$ the size of their overlap, *i.e.*, the number of elements in both π and τ.

Definition 1. *For two partial rankings τ and σ with $n[\tau, \sigma] \geq 2$, the Kendall-τ distance between π and σ is defined as $D(\tau, \sigma) = |\{(i, j) : \tau(i) < \tau(j), \text{but } \sigma(i) > \sigma(j), \forall i, j \in N_\pi \cap N_\sigma\}|$. The coherence of τ and σ is defined as*

$$\Phi(\tau, \sigma) = n[\tau, \sigma] \left(1 - \frac{D(\tau, \sigma)}{\binom{n[\tau, \sigma]}{2}} \right).$$

When $n[\tau, \sigma] \leq 1$, we define the coherence $\Phi(\tau, \sigma) = 0$.

Definition 2. *For a collection of partial rankings $\Pi = (\tau_1, \tau_2, \cdots, \tau_K)$ and a certain (partial) ranking π with respect to $N = N_{\tau_1} \cup \cdots \cup N_{\tau_K}$, the total coherence between π and $\Pi = (\tau_1, \tau_2, \cdots, \tau_K)$ is*

$$\Phi(\pi; \Pi) = \sum_{s=1}^{K} \Phi(\pi, \tau_s) = \sum_{s=1}^{K} n[\pi, \tau_s] \left(1 - \frac{D(\pi, \tau_s)}{\binom{n[\pi, \tau_s]}{2}} \right).$$

The coherence aggregation problem (CAP) is to find a (partial) ranking of specified length L ($L \leq n$) with respect to $N = N_{\tau_1} \cup \cdots \cup N_{\tau_K}$, which maximizes the total coherence $\Phi(\pi, \Pi)$ over all rankings π of length L. Especially, when all the rankings concerned are top-d rankings, the problem is called top-d CAP.

When the profile Π is clear from the context we will denote $\Phi(\pi, \Pi)$ by $\Phi(\pi)$. We note that the coherence aggregation problem is equivalent to Kemeny aggregation problem in a weighted version, where the weight of each given ranking is determined by the overlap with the final ranking.

There are many results on the complexity of rank aggregation problems. Bartholdi, et al.[3], proved that the Kemeny aggregation problem is NP-hard for an unbounded number of complete rankings. Their proof can also derive the proof of NP-hardness for CAP for an unbounded number of partial rankings with

unbounded length. On the other hand, Dwork, et al.[5], discussed the hardness in the setting of interest in meta-search: many alternatives and very few voters. They showed that computing a Kemeny optimal ranking for a collection of given rankings $\Pi = (\tau_1, \tau_2, \cdots, \tau_m)$ is still NP-hard for any fixed even $m \geq 4$. Their result derives directly the NP-hardness of the CAP for all integer $m \geq 4$, since odd number of partial rankings can be obtained from even number of complete rankings by splitting one complete ranking into two partial rankings.

Theorem 1. *The CAP and top-d CAP for a collection of K partial rankings, for integer $K \geq 4$, are all NP-hard.*

3 Heuristic Algorithm

In this section, we discuss a practice heuristic algorithm for top-d CAP. Given a profile of rankings $\Pi = (\tau_1, \tau_2, \cdots, \tau_m)$. Obviously, for any subset $S \subset N_{\tau_1} \cup \cdots \cup N_{\tau_m} = N$ of size $|S| = d$, the problem of finding a ranking on S maximizing the total coherence is equivalent to complete CAP on the profile $\Pi_S = (\tau_1|_S, \cdots, \tau_m|_S)$, where $\tau_i|_S$ $(i = 1, 2, \cdots, m)$ is the restriction of ranking τ_i on S. For any ranking σ and its reverse σ^r on S, we have $\Phi(\sigma)+\Phi(\sigma^r) = \sum_{l=1}^{m} |N_{\tau_l} \cap S|$. It implies that the optimal coherence value of the top-d CAP is no less than $\frac{1}{2} \max_{|S|=d} \sum_{l=1}^{m} |N_{\tau_l} \cap S|$. Thus, a simple 2-approximation algorithm can be obtained as follows.

Heuristic Algorithm

Step 1. Defining the degree $d(i)$ of element $i \in N$ as $d(i) = |\{l : i \in N_{\tau_l},\ \forall l = 1, 2, \cdots, m\}|$, and choosing d elements with the largest degrees. This derives a subset $S^* \subset N$ which maximizes $\sum_{l=1}^{m} |N_{\tau_l} \cap S|$ over all subsets of size d.

Step 2. Giving any ranking σ and its reversal σ^r on the subset S^* obtained in Step 1, and choosing one of them with the larger coherence value as the approximate solution of the problem.

Remark 1. When S^* is specified, we can construct a better ranking on S^* using the heuristic algorithm presented in [4].

4 Polynomial Time Approximation Scheme

In this section, we mainly discuss the following *restricted top-d CAP*: For a collection of partial rankings $\tau_1, \tau_2, \cdots, \tau_K$ of the same length d, where K is an integer indifference of d, the objective is to find a final partial ranking π of length d defined on $N = N_{\tau_1} \cup \cdots \cup N_{\tau_m} = \{1, 2, \cdots, n\}$, such that π maximizes the total coherence $\Phi(\pi) = \sum_{i=1}^{m} \Phi(\pi, \tau_i)$ under the constraints $|N_\pi \cap N_{\tau_i}| \geq \alpha d$ $(\forall i = 1, 2, \cdots, m)$ for given $0 < \alpha < 1$. These constrains imply that the final ranking must include sufficient information in every given partial ranking.

Arora, et al.[1,2], presented a unified approach, exhaustive sampling and transforming polynomial constraints into linear constraints, for developing into approximation algorithms for polynomial integer programs with 'smooth' coefficients. They designed PTASs for some 'smooth' dense subcases of many well known NP-hard arrangement problems, including minimum linear arrangement, d-dimensional arrangement, betweenness, maximum acyclic subgraph, etc. Their technique underlines our PTAS for restricted top-d CAP. We further exploit Arora's technique of exhaustive sampling, and it enables us to estimate the non-constant coefficients of the objective function and to transform the quadratic program into a linear program simultaneously. For briefness, we omit the proofs of the lemmas in this section.

For any ranking π of length d and its reversal π^r with respect to subset $T \subseteq N$, $\Phi(\pi) + \Phi(\pi^r) = \sum_{i=1}^{m} |T \cap N_{\tau_i}| \geq d$, so the optimal value of the top-d CAP is no less than $d/2$. Therefore to obtain an optimal ranking with at least the value $(1-\gamma)$ times the optimum, where $\gamma > 0$ is arbitrary, it suffices to find a ranking whose value is within an additional factor of ϵd from the optimal value of the optimal partial ranking for a suitable $\epsilon > 0$.

Let ϵ be a given small positive, and $t = c/\epsilon$ for some suitable large constant $c > 0$. Here we assume for simplicity that d is a multiple of t. Construct $t+1$ sequential groups $I_0, I_1, \cdots, I_t$. A *placement* is a mapping $g : N \to \{0, 1, \cdots, t\}$ from the set N to the set of groups $I_0, I_1, \cdots, I_t$. It is *proper* if it maps $n-d$ elements to I_0 and maps d/t elements of N to each group I_j $(j = 1, \cdots, t)$. Given a placement $g : N \to \{0, 1, \cdots, t\}$, denote

$$n_i^g = |\{j \in N : g(j) > 0\}|,$$

$$\omega_i^g = \begin{cases} 2/(n_i^g - 1) & n_i^g > 1 \\ 0 & n_i^g \leq 1 \end{cases} \quad i = 1, 2, \cdots, m.$$

The value of a placement g, denoted by $\phi(g)$, is defined as

$$\phi(g) = \sum_{s=1}^{m} \omega_s^g \ |\{(i,j) : \tau_s(i) < \tau_s(j) \text{ and } 0 < g(i) < g(j)\}|.$$

Note that every partial ranking of length d induces a proper placement, in which the elements not ranked are placed in the group I_0. Our placement problem is defined as: For a collection of partial ranking $\tau_1, \tau_2, \cdots, \tau_m$ with the same length d, we are to find a proper placement g which maximize the value $\phi(g)$ under the constrains $n_i^g \geq \alpha d$ $(\forall i = 1, 2, \cdots, n)$. The following result gives the relationship between approximate optimal solutions of CAP and placement problem.

Lemma 1. *Let π^* be an optimal partial ranking, g^* be its induced placement. If g is the placement induced by a partial ranking π such that $\phi(g) \geq \phi(g^*) - \epsilon' d$, then*

$$\Phi(\pi) \geq \Phi(\pi^*) - \epsilon d,$$

where $\epsilon' = (1 - 3m/c)\epsilon$ for given $\epsilon > 0$.

Therefore, finding an approximate optimal solution to the restricted top-d CAP can be reduced to the problem of finding a proper placement within an additive factor of $\epsilon' d$ from the corresponding optimal placement. The placement problem can be formulated as a quasi-quadratic arrangement programming:

$$\text{Max} \sum_{s=1}^{m} \sum_{ikjl} c^s_{ikjl} x_{ik} x_{jl}$$

$$\text{s.t.} \begin{cases} \sum_{i=1}^{n} x_{i0} = n - d \\ \sum_{i=1}^{n} x_{ik} = n/t \quad k = 1, 2, \cdots, t \\ \sum_{k=0}^{t} x_{ik} = 1 \quad i = 1, 2, \cdots, n \\ \sum_{i \in N_{\tau_s}} \sum_{k>0} x_{ik} \geq \alpha d \; s = 1, 2, \cdots, m \\ x_{ik} = 0, 1 \quad i = 1, 2, \cdots, n; \; k = 1, 2, \cdots, t \end{cases}$$

Here the coefficients c^s_{ikjl}'s are not constants, they are determined by the values of variables x_{ik}'s. For a feasible solution x to this problem, which corresponds to a placement g, the coefficients in the objective function are:

$$c^s_{ikjl} = \begin{cases} \omega^g_s & \text{if } \tau_s(i) < \tau_s(j) \text{ and } 0 < k < l \\ 0 & \text{otherwise} \end{cases}$$

Let g^* be the optimal placement, and denote

$$\hat{e}^s_{ik} = \textstyle\sum_{jl} \hat{c}^s_{ikjl} \; g^*_{jl} = \omega^{g^*}_s |\{j \in N_{\tau_s} : \tau_s(i) < \tau_s(j), \\ g^*(j) > k > 0\}| = \omega^{g^*}_s \hat{f}^s_{ik}.$$

We use the unified framework of exhaustively sampling presented in [1,2] to estimate $\omega^{g^*}_s$ and $\hat{f}^s_{ik}$ simultaneously. However, since the size of the overlap of each given partial ranking and the final ranking may be quite different, the coefficients do not satisfy the 'smoothness' condition required in Arora's framework. Thus, we extend Arora's method by making random sampling independently for each partial ranking and estimating the coefficients related $\tau_1, \tau_2, \cdots, \tau_m$ separately.

Our procedure of exhaustively sampling is as follows. We randomly pick with replacement a multi-set T_s of $O(\log d/\delta^2)$ elements (where δ is a sufficiently small fraction of ϵ' which we will determine later) from the set N_{τ_s} $(s = 1, \cdots, m)$ respectively, and thus we choose randomly a multi-set $T = T_1 \cup \cdots \cup T_m$ of size $|T| = O(\log d)$.

Especially, since the final ranking should satisfy the constraints $|N_\pi \cap N_{\tau_i}| \geq \alpha d$ $(\forall i = 1, 2, \cdots, m)$, we need only enumerate all possible function $h : T \to \{0, 1, \cdots, t\}$, which satisfies that

$$\forall s = 1, 2, \cdots, m : \; |\{i \in T_s : h(i) > 0\}| \geq (\alpha - \delta)|T_s|.$$

For each such function, we solve a linear program $\mathcal{M}_h$ described below, and round the (fractional) optimal solution to construct a proper placement if its

feasible solution set is not empty. Among all these placements, we pick up one with maximum value. When the function h we considered is the same as h^* which is the restriction of an optimal placement g^* on T, the placement g we get from the linear program $\mathcal{M}_h$ will satisfy

$$\phi(g) \geq \phi(g^*) - \epsilon' n$$

with high probability, over the random choice of T.

Let g be an arbitrary placement, and h be the restriction of g on T. For simplicity, we will identify h with its restrictions on T_s's $(s = 1, 2, \cdots, m)$ in the rest of this section. Making use of the placement of the sampled elements, we estimate ω_s^g and $\hat{f}_{ik}^s$ for each partial ranking τ_s using ϖ_s and f_{ik}^s:

$$\begin{aligned}
\rho_s &= \frac{d}{|T_s|}|\{j \in T_s : h(j) > 0\}|; \\
\varpi_s &= \frac{2}{\rho_s - 1}; \\
f_{ik}^s &= \frac{d}{|T_s|}|\{j \in T_s : \tau_s(i) < \tau_s(j) \text{ and } h(j) > k > 0\}|.
\end{aligned}$$

Lemma 2. *Pick uniformly at random with replacement a multi-set T_s of $O(\log d/\delta^2)$ elements from N_{τ_s}. Let g be a placement, and h be the restrictions of g on T_s. Then with high probability (over the choice of sample T_s), we have*
(1) $|\rho_s - n_s^g| \leq \delta d$ and $|\varpi_s - \omega_s^g| \leq \alpha\delta/d$, where α is a positive constants;
(2) $|f_{ik}^s - \hat{f}_{ik}^s| \leq \delta d$.

Consider the following linear program $\mathcal{M}_h$:

$$\text{Max} \quad Z(x) = \sum_{s=1}^{m}\Big(\sum_{i\in N_{\tau_s}}\sum_{k=1}^{t}\varpi_s f_{ik}^s x_{ik}\Big)$$

$$\text{s.t.} \quad \begin{cases}
\sum_{i=1}^{n} x_{i0} = n - d & \\
\sum_{i=1}^{n} x_{ik} = d/t & k = 1, 2, \cdots, t \\
\sum_{k=0}^{t} x_{ik} = 1 & i = 1, 2, \cdots, n \\
\sum_{i\in N_{\tau_s}}\sum_{k>0} x_{ik} \geq \alpha d & s = 1, 2, \cdots, m \\
|\sum_{i\in N_{\tau_s}}\sum_{k>0} x_{ik} - \rho_s| \leq \delta d & s = 1, 2, \cdots, m \\
\left|\sum_{j:\tau_s(i)<\tau_s(j)}\ \sum_{l>k>0} x_{jl} - f_{ik}^s\right| \leq \delta d & s = 1, 2, \cdots, m;\ i \in N_{\tau_s}; \\
& k = 1, 2, \cdots, t \\
0 \leq x_{ik} \leq 1 & i = 1, 2, \cdots, n;\ k = 1, 2, \cdots, t
\end{cases}$$

We solve $\mathcal{M}_h$ for every possible assignment h. Let x^h be the optimal (fractional) solution for $\mathcal{M}_h$. We round x_{ik}^h using randomized rounding techniques of Raghavan and Thompson [9] to obtain a placement $\tilde{r}$ and corresponding proper

placement r^h as follows: (1) for each element i, independently take $\tilde{r}(i) = k$ with probability x^h_{ik}; (2) construct a proper placement r^h from $\tilde{r}$ by moving elements from groups with more than n/t elements ($n - d$ elements for I_0)assigned to them to groups with less than n/t elements ($n - d$ elements for I_0) assigned to them arbitrarily.

Let h^* be the restriction of the optimal placement g^* to the sample subset T. Let r^* be the proper placement which we constructed by rounding the optimal solution x^* of $\mathcal{M}_{h^*}$. The following lemma gives the relation between the optimal value of the linear programming $Z(x^*)$ and the value of corresponding proper placement $\phi(r^*)$.

Lemma 3. *Let r^* be the proper placement constructed from the optimal (fractional) solution x^* of $\mathcal{M}_{h^*}$, where h^* is the restriction of an optimal placement g^* to T. Then there is a constant $\beta > 0$ such that with high probability*

$$\phi(r^*) \geq Z(x^*) - \beta\delta d.$$

Following from Lemma 2, we have with high probability that g^* is a feasible solution to $\mathcal{M}_{h^*}$. Hence

$$\begin{aligned} Z(x^*) \geq Z(g^*) &= \sum_{s=1}^{m} \sum_{i \in N_{\tau_s}} \sum_{k=1}^{t} \varpi_s f^s_{ik} g^*_{ik} \\ &\geq \sum_{s=1}^{m} \sum_{i \in N_{\tau_s}} \sum_{k=1}^{t} \omega^{g^*}_s f^s_{ik} g^*_{ik} - \sum_{s=1}^{m} \sum_{i \in N_{\tau_s}} \sum_{k=1}^{t} |\varpi_s - \omega^{g^*}_s| f^s_{ik} g^*_{ik} \\ &\geq \phi(g^*) - \beta' \delta d, \end{aligned}$$

where the coefficient $\beta' > 0$ is a constant. By choosing $\delta = \epsilon'/(\beta + \beta')$ and also by Lemma 3, we have

$$\phi(r^*) \geq Z(x^*) - \beta\delta d \geq \phi(g^*) - (\beta + \beta')\delta d \geq \phi(g^*) - \epsilon' d.$$

On the other hand, in the linear program $\mathcal{M}_h$, it is demanded that

$$\sum_{i \in N_{\tau_s}} \sum_{k>0} x_{ik} \geq \alpha d.$$

So with high probability, the placement r^h we constructed satisfies

$$n^{r^h}_s \geq \alpha d - O(\sqrt{d \log d}) = (\alpha - o(1))d.$$

Since r^* is a candidate for our chosen placement r^h, and we choose the placement with maximum value which is no less than the value of r^*, therefore, we obtain the desired result.

Theorem 2. *There is a randomized polynomial time algorithm (PTAS) that approximately solves the restricted top-d CAP in the following sense. Suppose*

the partial ranking π^* is an optimal solution of the restricted top-d CAP. Then for any fixed $\epsilon > 0$, in time $d^{O(1/\epsilon^2)}$ the algorithm finds a partial ranking π of length d satisfying the constraints $|N_\pi \cap N_{\tau_i}| \geq (\alpha - o(1))d$ $(\forall i = 1, 2, \cdots, m)$ and

$$\Phi(\pi) \geq \Phi(\pi^*) - \epsilon d.$$

5 Conclusion and Further Work

Considering the distinct features in the context of meta-search on the web, we have developed a new rank aggregation method based on the criterion of Coherence. The practical heuristic algorithm with performance ratio 2 proposed for the top-d CAP is also suitable for the general coherence aggregation problems. We also presented a PTAS for the restricted top-d CAP. Our algorithm extends the general technique of Arora, et al.[1,2], for approximating 'smooth' polynomial integer programs to more complicated quasi-quadratic programs, in which the coefficients of objective function are not constants. However, the exhaustive random sampling technique that underlies our algorithm no longer suffices for the general CAP without the restricted condition, since the additive approximation of the coefficients in the objective function is not good enough in general case. This needs some other approximation method to solve it. Besides the Kendall-τ distance and coherence, other metrics in social choice theory are also worth of further exploration with the algorithmic approach.

Acknowledgments

This work is partly supported by NCET (No.05-0598) and NSFC (No.10771200). Dr Shanfeng Zhu is supported by the National Nature Science Foundation of China (No. 60903076) and the Shanghai Committee of Science and Technology, China (No.08DZ2271800 and 09DZ2272800).

References

1. Arora, S., Frieze, A., Kaplan, H.: A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. Mathematical Programming, Ser. A 92, 1–36 (2002)
2. Arora, S., Karger, D., Karpinski, M.: Polynomial-time approximation schemes for dense instances of NP-hard optimization problems. Journal of Computer and System Sciences 58, 193–210 (1999)
3. Barthelemy, J.P., Guenoche, A., Hudry, O.: Median linear orders: Heuristics and a branch and bound algorithm. European Journal of Operational Research 42, 313–325 (1989)
4. Chin, F.Y.L., Deng, X., Fang, Q., Zhu, S.: Approximate and dynamic rank aggregation. Theoretical Computer Science 352, 409–424 (2004)
5. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. WWW10, pp. 613–622 (2001)

6. Gravano, L., Chang, C., Garcia-Molina, H., Paepcke, A.: STARTS: Stanford proposal for internet meta-searching. ACM SIGMOD, Tucson, 207–218 (May 1997)
7. Hoelscher, C.: How Internet Experts Search for Information on the Web. In: The World Conference of the World Wide Web, Internet, and Intranet, Orlando, FL (1998)
8. Jansen, B.J., Spink, A., Saracevic, T.: Real life, real users, and real needs: a study and analysis of user queries on the web. Information Processing and Management 36, 207–227 (2000)
9. Raghavan, P., Thompson, C.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica 7, 365–374 (1987)
10. Silverstein, C., Henzinger, M., Marais, H., Moricz, M.: Analysis of a very large altavista query log. Technical Report SRC 1998-014, Digital Systems Research Center (1998)

Minimum Common String Partition Revisited

Haitao Jiang[1,2], Binhai Zhu[1], Daming Zhu[2], and Hong Zhu[3]

[1] Department of Computer Science, Montana State University,
Bozeman, MT 59717-3880, USA
{htjiang,bhz}@cs.montana.edu
[2] School of Computer Science and Technology, Shandong University, Jinan, China
dmzhu@sdu.edu.cn
[3] College of Software, East China Normal University, Shanghai, China
hzhu@sei.ecnu.edu.cn

Abstract. Minimum Common String Partition (MCSP) has drawn much attention due to its application in genome rearrangement. In this paper, we investigate three variants of MCSP: $MCSP^c$, which requires that there are at most c symbols in the alphabet; d-MCSP, which requires the occurrence of each symbol to be bounded by d; and x-balance MCSP, which requires the length of blocks not being x away from the average length. We show that $MCSP^c$ is NP-hard when $c \geq 2$. As for d-MCSP, we present an FPT algorithm which runs in $O^*((d!)^k)$ time. As it is still unknown whether an FPT algorithm only parameterized on k exists for the general case of MCSP, we also devise an FPT algorithm for the special case x-balance MCSP parameterized on both k and x.

1 Introduction

String comparison has drawn a lot of attention due to its applications in computational biology, text processing and compression. In this paper, we revisit the Minimum Common String Partition (MCSP) problem which has a close relation to the genome rearrangement problems such as Edit distance, Sorting by Reversals and Transpositions, etc.

A partition P of a string X is a sequence $P = \langle P_1, P_2, \ldots, P_m \rangle$ of strings whose concatenation is equal to X, that is $P_1 P_2 \ldots P_m = X$. The strings P_i are called the blocks 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.

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 satisfy this condition. Clearly, this is a sufficient and necessary condition for X and Y to have a common string partition. 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

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 45–52, 2010.

are several variants of MCSP. The restricted version where each letter occurs at most d times in each input string, is denoted by d-MCSP. Another important version where the input strings are over an alphabet with size bounded by c, is abbreviated as $MCSP^c$. We also introduce a feasible version where the resulting blocks in the partition is designated to have nearly even length, we call it x-balance MCSP. For the x-balance MCSP problem, if the optimum solution has k blocks, the length of each block ranges from $n/k - x$ to $n/k + x$. The signed minimum common string partition problem (SMCSP) is also a variant of MCSP in which each letter of the two input strings is given a "+" or "-" sign. For a string S with signs, let $-S$ denote the reverse of S, with each letter sign flipped. The common partition has a little bit of difference from the original where $P_i = Q_{\sigma_i}$ or $P_i = -Q_{\sigma_i}$.

Related work

The proble d-MCSP is well studied. 2-MCSP (and therefore MCSP) is NP-hard; moreover, APX-hard [8]. Several approximation algorithms are known for the problem [8,10]. 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 multiplicative factor 2 of each other. Kolman [12] devised an $O(d^2)$-approximation algorithm running in $O(n)$ time for SRDD.

Chrobak et al. [3] 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 $O(n^{0.67})$. The same bounds apply for SMCSP. Kaplan and Shafrir [9] improved the lower bound to $\Omega(n^{0.46})$ when the input strings are over an alphabet of size $O(\log n)$. Kolman [11] described a simple modification of the greedy algorithm; the approximation ratio of the modified algorithm is $O(p^2)$ for p-MCSP. Christie and Irving [2] proved that the problem of computing (unsigned) reversal distance is NP-hard for binary strings, it turns out that this problem has some connection to $MCSP^c$.

On 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].

Our Contribution

We prove that the problem $MCSP^c$ is NP-complete when $c \geq 2$. For d-MCSP, we introduce an equivalence condition for the common partition and present an FPT algorithm for that problem with running time $O^*((d!)^k)$. When the MCSP solution is supposed to be x-balance, we devise an FPT algorithm with parameter k and x which runs in $O((2x)^k k!n))$ time.

2 Preliminaries

As aforementioned, we recall the formal definition of MCSP.

Minimum Common String Partition
Input: two strings X, Y over an alphabet Σ
Question: Can X have a partition P and Y have a partition Q such that Q is a permutation of P ?

We say that the elements from Σ *occur* or *appear* in X and Y. A specific occurrence of some element is called a *symbol*. The strings in the partition are called blocks. There is a *break* or *cut* between two consecutive blocks in X or Y respectively.

A *duo* is a substring of length two. A *specific duo* is an occurrence of a duo in X or Y. Two specific duos are continuous if they form a substring of length three. A match is a pair $(a_i a_{i+1}, b_j b_{j+1})$ of specific duos, one from X and the other one from Y, such that $a_i = b_j$ and $a_{i+1} = b_{j+1}$. We also say that the pair of specific duos are matched if they form a match. Two matches form a *conflict* if they cannot be realized at the same time.

An FPT (Fixed-Parameter Tractable) algorithm for an optimization problem Π with optimal solution value k is an algorithm which solves the problem in $O(f(k)n^c)$ time, where f is any function only on k, n is the input size and c is some fixed constant not related to k. For convenience we also say that Π is in FPT. More details on FPT algorithms can be found in [5,6].

It is open whether an FPT algorithm exists for the general *MCSP* with k being the unique parameter, so we try to consider variants of it by the use of additional parameters.

This paper is organized as follows. In Section 2, we present the NP-completeness of $MCSP^c$. In Section 3, we present FPT algorithms for the two special cases. In Section 4, we conclude the paper with several open questions.

3 Hardness for $MCSP^c$

In this section, we prove that $MCSP^c$ is NP-complete when $c \geq 2$ by a reduction from 3-PARTITION [7]. Firstly, we go over the formal definition of 3-PARTITION.

3-PARTITION
Input: Positive integers n and B, and positive integers set $A = \{a_1, a_2, \ldots, a_{3n}\}$, with $B/4 < a_i < B/2$ and $\sum_{a_i \in A} a_i = nB$.
Question: Can A be partitioned into n disjoint sets $S_1, S_2, \ldots, S_n$ such that, for $1 \leq i \leq n$, $\sum_{a_j \in S_i} a_j = B$.

The problem 3-PARTITION is strongly NP-hard: that is, there is a polynomial $p(n)$ such that it is still NP-hard when all the a_i's are at most $p(n)$. Our reduction is polynomially bounded for instances of this type.

Given an instance of 3-PARTITION with integers (weights) $a_1, a_2, \ldots, a_{3n}$, we construct two strings X, Y for an instance of $MCSP^2$ as follows

$$X = 11110^{a_1}11110^{a_2}\cdots 11110^{a_{3n}}$$

$$Y = (0^B1)^n1^{11n}$$

For the sake of clarity, we first make the following claims.

Claim. Any block in the common string partition has the form 0^p1, 10^p, 0^p or 1^q where $p, q \geq 1$.

Proof. Suppose to the contrary that there is a common block H with the form $0^p1^q0^r\cdots$, then from the format of X and Y, q is either equal to one or equal to four. But H cannot be a substring of X when $q = 1$ and cannot be a substring of Y when $q = 4$. Similarly, any block in the common partition will not have the form $1^q0^p1^s\cdots$. Consequently, all the common blocks must be of the form 0^p1, 10^p, 0^p and 1^q, as there is only one 1 between two 0's in Y. The claim holds. □

Claim. The interior '11' from '1111' in X is matched to '11' from 1^{11n+1} in Y.

Proof. From the first claim we conclude that, if a common block contains 0, then it contains at most one 1. So the interior two 1's form '1111' must be in the common blocks containing only 1's. Since there are no consecutive 1's in Y besides the substring 1^{11n+1}, the claim is certainly true. □

Theorem 1. *$MCSP^2$ is NP-complete.*

Proof. We prove that the 3-PARTITION has an precise partition if and only if X, Y have a common partition with $6n$ blocks.

On the necessary side, assume that $S_1, S_2, \ldots, S_n$ satisfy $\sum_{a_j \in S_i} a_j = B$, for all i. For any $S_i = \{a_p, a_q, a_r\}$, we obtain three blocks form X: $0^{a_p}, 0^{a_q}, 0^{a_r}1$ and one block from Y: 0^B1. Obviously, the block from Y can be divided into three blocks corresponding to the blocks from X. Then there remains $2n$ blocks of the form '1111' and n blocks of the form '111' in X and all these blocks are separated (not adjacent). The unique block left in Y is 1^{11n+1}. Consequently, we obtain a common partition of X, Y with $6n$ blocks.

On the sufficient side, from the two claims, we can see that all the 0^{a_i} and the interior substring '11' of '1111' are in different blocks in the common partition. So there are at least $6n$ blocks in any common partition. If the number of blocks is exactly $6n$, then every block of the form 0^B in Y is matched to exactly three blocks of the form 0^{a_i}, which means it is a yes 3-PARTITION instance. □

Since we can construct an instance of $MCSP^c$ when $c > 2$ from an instance if $MCSP^2$ by adding the same substring composed of symbols other than those in $MCSP^2$ at the ends of the two input strings, we can easily obtain the following corollary.

Corollary 1. *$MCSP^c$ is NP-complete for $c \geq 2$.*

4 The FPT Algorithms

In this section, we design FPT algorithms for two variants of MCSP. As it is still unknown whether MCSP has an FPT algorithm parameterized only on k, we hope these algorithms could help shed light on answering this open question. On the other hand, these two variants are closely related to genome rearrangement problems, hence are meaningful practically.

4.1 On *d-MCSP*

In this subsection, we describe the FPT algorithm for *d-MCSP*.

First, let us recall the definition of duos and matches. Note that two matches with the continuous specific duos in one string are not in conflict if and only if these duos are matched to two continuous specific duos in the other string.

Lemma 1. *Each duo appears the same number of times in X and Y in the common partition.*

Proof. Otherwise, there must be such a specific duo that is unmatched in the partition. □

Lemma 2. *If each duo appears the same number of times in a partition of X and a partition of Y, then two matches are conflict if and only if the specific duos are continuous in one string and not continuous in the other.*

Proof. A conflict match means that there exists a symbol which is matched to two occurrences of that symbol in the matches. Clearly in this case the specific duos cannot be continuous in both strings. □

Lemma 3. *Given a pair of partition $\pi = (P, Q)$ for X and Y, if all specific duos are matched and all matches are not conflict then π is a common partition.*

Proof. For any block $G = g_1 g_2 \cdots g_m$ in P, since all the specific duos $g_i g_{i+1}$ are matched, there must be matches $(g_i g_{i+1}, h_j h_{j+1})$ and $(g_{i+1} g_{i+2}, f_k f_{k+1})$ that can be realized at the same time. Since $g_i g_{i+1}$ and $g_{i+1} g_{i+2}$ are continuous duos, $h_j h_{j+1}$ and $f_k f_{k+1}$ should be continuous too; that is, h_{j+1} and f_k are the same symbol. As the above analysis holds for all i, we can see that there is a block $H = h_1 h_2 \cdots h_m$ in Q such that $G = H$. The lemma holds. □

Our FPT algorithm just follows from the three lemmas. The rough idea is to cut redundant specific duos such that all duos appear the same number of times in X and Y, then cut one of the continuous duos that corresponds to two conflict matches. In the algorithm, we use arrays C and D indexed by duos to store the number of occurrence of each duo in X and Y respectively. That is, $C_{ab} = r$ means ab appears r times in X.

Theorem 2. *Algorithm Cut-Duos runs in $O^*((d!)^k)$ time.*

Proof. From the algorithm we branch on step 2, 3 and 5. As for the cost of the algorithm, Step 2 has recurrence relation

$$f(k) = \binom{r}{s}(s!)f(k-(r-s)).$$

Step 3 has a similar recurrence relation. Step 5 has recurrence relation

$$f(k) = 2f(k-1).$$

Since $r, s \leq d$ and $f(k)$ achieves its maximum value when $r - s = 1$, so $f(k) = O^*((d!)^k)$. □

Algorithm. *Cut-Duos*
Input: *two strings X, Y such that each symbol appears at most d time in each. string*
Output: *A common partition (P, Q)*
1 Compute the number of occurrence of each duo in X and Y, and store them in arrays C and D.
2 For every duo ab in X, if $r = C_{ab} > D_{ab} = s$
2.1 Choose $r - s$ ab's in X, cut them.
2.2 Compute matches between the remaining ab's in X and Y.
3 For every duo ab in Y, if $r = C_{ab} < D_{ab} = s$
3.1 Choose $s - r$ ab's in Y, cut them.
3.2 Compute matches between the remaining ab's in X and Y.
4 For two conflict matches (ab, ab) and (bc, bc) with the common symbol b in X and Y.
5 Choose one duo from ab and bc, cut it in both X and Y.
6 Return the remaining blocks in X and Y respectively as the common partition (P, Q).

4.2 On x-Balance MCSP

In this subsection, we deal with the x-balance MCSP problem. For the specific applications in genome arrangement, due to the relation between MCSP and genome rearrangement, each block corresponds to a gene and each gene should have roughly the same gene content, so the blocks are supposed to have nearly even lengths, possibly with some small deviation. Assume that the optimal solution size is k, then we can see that every block is of some length between $(n/k) + x$ and $(n/k) - x$.

The rough idea of the FPT algorithm is to cut the input string into blocks of length ranging from $(n/k) - x$ to $(n/k) + x$, then compute all possible matches between the resulting k blocks. In the algorithm, assume that the length of the ith block is $L(i)$. When we refer to *positions* from integer u to v $(u < v)$ in X, we mean the set of duos $\{X_uX_{u+1}, X_{u+1}X_{u+2}, \ldots, X_{v-1}X_v\}$.

Theorem 3. *Algorithm Cut-Depending-on-Length runs in $O((2x)^k k!n)$ time.*

Proof. From the algorithm we branch on step 2, 3 and 4. Step 2 runs $(2x)^{k-1}$ times. Step 3 has a similar running time. Step 4 has $k!$ possibilities. Step 5 runs in $O(n)$ time. So the total running time of the algorithm is $O((2x)^k k!n))$. □

We comment that Algorithm Cut-Depending-on-Length works for other varieties of MCSP whenever they are x-balance.

5 Concluding Remarks

In this paper, we deal with some variants of MCSP. The first one, $MCSP^c$, is proved to be NP-complete when $c \geq 2$. For d-MCSP and x-balance MCSP, we devise FPT algorithms for them. The original MCSP problem seems difficult to solve with an FPT algorithm only on the parameter k, so some extra practical parameters which make the problem fixed-parameter tractable are meaningful. As d-MCSP has close relation to some specific genome rearrangement problem, we are currently seeking good approximations for the d-MCSP problem.

Algorithm. *Cut-Depending-on-Length*
Input: *an x-balance instance of MCSP*
Output: *A common partition (P,Q)*
1 Compute the scope of the length of blocks.
2 For input string X, for $1 \leq i \leq k-1$
2.1 Let S_0 be set of position from $n-(n/k+x)(k-i)$ to $n-(n/k-x)(k-i)$
2.2 Let S_1 be set of position from $\sum_{0\leq j<i} L(j) + (n/k-x)$ to $\sum_{0\leq j<i} L(j) + (n/k+x)$
2.3 Choose a duo from $S_0 \cap S_1$ and cut it.
3 For input string Y, for $1 \leq i \leq k-1$
3.1 Let S_0 be set of position from $n-(n/k+x)(k-i)$ to $n-(n/k-x)(k-i)$
3.2 Let S_2 be set of position from $\sum_{0\leq j<i} L(j) + (n/k-x)$ to $\sum_{0\leq j<i} L(j) + (n/k+x)$
3.3 Choose a duo from $S_0 \cap S_2$ and cut it.
4 Compute a one-to-one mapping between the resulting blocks in X and Y.
5 Check whether all the two blocks in the map are common.

Acknowledgments

This research is partially supported by NSF grant DMS-0918034, and by NSF of China under grant 60928006. We also thank anonymous reviewers for some constructive comments.

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. Christie, D.A., Irving, R.W.: Sorting strings by reversals and by transpositions. SIAM Journal on Discrete Mathematics 14(2), 193–206 (2001)
3. 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)

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, A., Kolman, P., Zheng, J.: Minimum common string partitioning problem: Hardness and approximations. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 473–484. Springer, Heidelberg (2004); also in: The Electronic Journal of Combinatorics 12 (2005), paper R50
9. Kaplan, H., Shafrir, N.: The greedy algorithm for edit distance with moves. Inf. Process. Lett. 97(1), 23–27 (2006)
10. 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)
11. 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)
12. Kolman, P., Walen, T.: Approximating reversal distance for strings with bounded number of duplicates. Discrete Applied Mathematics 155(3), 327–336 (2007)

Inapproximability of Maximal Strip Recovery: II*

Minghui Jiang

Department of Computer Science, Utah State University, Logan, UT 84322, USA
mjiang@cc.usu.edu

Abstract. Maximal Strip Recovery (MSR) is an optimization problem proposed by Zheng, Zhu, and Sankoff for reliably recovering syntenic blocks from genomic maps in the midst of noise and ambiguities. Given d genomic maps as sequences of gene markers, the objective of MSR-d is to find d subsequences, one subsequence of each genomic map, such that the total length of syntenic blocks in these subsequences is maximized. In our recent paper entitled "Inapproximability of Maximal Strip Recovery" in ISAAC 2009, we proved that MSR-d is APX-hard for any constant $d \geq 2$, and presented the first explicit lower bounds for approximating MSR-2, MSR-3, and MSR-4, even for the most basic version of the problem in which all markers are distinct and appear in positive orientation in each genomic map. In this paper, we present several further inapproximability results for MSR-d and its variants CMSR-d, δ-gap-MSR-d, and δ-gap-CMSR-d. One of our main results is that MSR-d is NP-hard to approximate within $\Omega(d/\log d)$ even if all markers appear in positive orientation in each genomic map. From the other direction, we show that there is a polynomial-time $2d$-approximation algorithm for MSR-d even if d is not a constant but is part of the input.

1 Introduction

In comparative genomic, the first step of sequence analysis is usually to decompose two or more genomes into syntenic blocks that are segments of homologous chromosomes. For the reliable recovery of syntenic blocks, noise and ambiguities in the genomic maps need to be removed first. A genomic map is a sequence of gene markers. A gene marker appears in a genomic map in either positive or negative orientation. Given d genomic maps, *Maximal Strip Recovery* (MSR-d) is the problem of finding d subsequences, one subsequence of each genomic map, such that the total length of strips of these subsequences is maximized [20,9]. Here a *strip* is a maximal string of at least two markers such that either the string itself or its signed reversal appears contiguously as a substring in each of the d subsequences in the solution. Without loss of generality, we can assume that all markers appear in positive orientation in the first genomic map.

For example, the two genomic maps (the markers in negative orientation are underlined)

1	2	3	4	5	6	7	8	9	10	11	12
$\underline{8}$	$\underline{5}$	$\underline{7}$	$\underline{6}$	4	1	3	2	$\underline{12}$	$\underline{11}$	$\underline{10}$	9

* Supported in part by NSF grant DBI-0743670.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 53–64, 2010.

have two subsequences

$$\begin{array}{cccccccccccc} 1 & 3 & & & 6 & 7 & 8 & & & 10 & 11 & 12 \\ \underline{8} & \underline{7} & \underline{6} & & & 1 & 3 & & & \underline{12} & \underline{11} & \underline{10} \end{array}$$

of the maximum total strip length 8. The strip $\langle 1, 3\rangle$ is positive and forward in both subsequences; the other two strips $\langle 6, 7, 8\rangle$ and $\langle 10, 11, 12\rangle$ are positive and forward in the first subsequence, but are negative and backward in the second subsequence. Intuitively, the strips are syntenic blocks, and the deleted markers not in the strips are noise and ambiguities in the genomic maps.

The problem MSR-2 was introduced by Zheng, Zhu, and Sankoff [20], and was later generalized to MSR-d for any $d \geq 2$ by Chen, Fu, Jiang, and Zhu [9]. For MSR-2, Zheng et al. [20] presented a potentially exponential-time heuristic that solves a subproblem of Maximum-Weight Clique. For MSR-d, Chen et al. [9] presented a $2d$-approximation based on Bar-Yehuda et al.'s fractional local-ratio algorithm for Maximum-Weight Independent Set in d-interval graphs [5]; the running time of this $2d$-approximation algorithm is polynomial if d is a constant. From the other direction, we recently proved that MSR-d for any constant $d \geq 2$ is APX-hard:

Theorem 1 (Jiang 2009 [13]). MSR-d *for any constant* $d \geq 2$ *is APX-hard. Moreover,* MSR-2*,* MSR-3*, and* MSR-d *for any constant* $d \geq 4$ *are NP-hard to approximate within any constants less than* $\frac{2320}{2319}$ (> 1.0004)*,* $\frac{474}{473}$ (> 1.0021)*, and* $\frac{285}{284}$ (> 1.0035)*, respectively, even if all markers are distinct and appear in positive orientation in each genomic map.*

We refer to [9,19,8] for more hardness results on MSR-d and its variants. In the biological context, a genomic map may contain duplicate markers as a paralogy set [20, p. 516], but such maps are relatively rare. Thus MSR-d without duplicates is the most useful version of MSR-d in practice. Also, all previous hardness proofs of MSR-d and its variants [9,19,8] rely on the fact that a marker may appear in a genomic map in either positive or negative orientation. A natural question is whether the complexity of the problem stays the same if all markers in the input genomic maps are in positive orientation. For these two reasons, our previous work [13] and our work in this paper investigate the most basic version of Maximal Strip Recovery in which all markers are distinct and appear in positive orientation in each genomic map.

The main result of this paper is the following theorem that strengthens Theorem 1 by an improved lower bound for MSR-4 and a new asymptotic lower bound for MSR-d:

Theorem 2. MSR-d *for any* $d \geq 2$ *is APX-hard. Moreover,* MSR-2*,* MSR-3*,* MSR-4*, and* MSR-d *are NP-hard to approximate within* 1.000431*,* 1.002114*,* 1.010661*, and* $\Omega(d/\log d)$*, respectively, even if all markers are distinct and appear in positive orientation in each genomic map.*

Recall that MSR-d admits a polynomial-time $2d$-approximation algorithm for any constant $d \geq 2$ [9]. Thus MSR-d for any constant $d \geq 2$ is APX-complete. Our following theorem gives a polynomial-time $2d$-approximation algorithm for MSR-d even if the number d of genomic maps is not a constant but is part of the input:

Theorem 3. *For any $d \geq 2$, there is a polynomial-time $2d$-approximation algorithm for* MSR-d *if all markers are distinct in each genomic map. This holds even if d is not a constant but is part of the input.*

Compare the upper bound of $2d$ in Theorem 3 and the asymptotic lower bound of $\Omega(d/\log d)$ in Theorem 2.

Maximal Strip Recovery [20,9] is a maximization problem. Wang and Zhu [19] introduced Complement Maximal Strip Recovery as a minimization problem. Given d genomic maps as input, the problem CMSR-d is the same as the problem MSR-d except that the objective is minimizing the number of deleted markers not in the strips, instead of maximizing the number of markers in the strips. A natural question is whether a polynomial-time approximation scheme may be obtained for this problem. Our following theorem shows that unless NP=P, CMSR-d cannot be approximated arbitrarily well:

Theorem 4. CMSR-d *for any $d \geq 2$ is APX-hard. Moreover,* CMSR-2*,* CMSR-3*,* CMSR-4*, and* CMSR-d *for any $d \geq 173$ are NP-hard to approximate within* 1.000625*,* 1.0101215*,* 1.0202429*, and $\frac{7}{6} - O(\log d/d)$, respectively, even if all markers are distinct and appear in positive orientation in each genomic map. If the number d of genomic maps is not a constant but is part of the input, then* CMSR-d *is NP-hard to approximate within any constant less than $10\sqrt{5} - 21 = 1.3606\ldots$, even if all markers are distinct and appear in positive orientation in each genomic map.*

Note the similarity between Theorem 2 and Theorem 4. In fact, our proof of Theorem 4 uses exactly the same constructions as our proof of Theorem 2. The only difference is in the analysis of the approximation lower bounds.

Bulteau, Fertin, and Rusu [8] recently proposed a restricted variant of Maximal Strip Recovery called δ-gap-MSR, which is MSR-2 with the additional constraint that at most δ markers may be deleted between any two adjacent markers of a strip in each genomic map. We now define δ-gap-MSR-d and δ-gap-CMSR-d as the restricted variants of the two problems MSR-d and CMSR-d, respectively, with the additional δ-gap constraint. Bulteau et al. [8] proved that δ-gap-MSR-2 is APX-hard for any $\delta \geq 2$, and is NP-hard for $\delta = 1$. We extend our proofs of Theorem 2 and Theorem 4 to obtain the following theorem on δ-gap-MSR-d and δ-gap-CMSR-d for any $\delta \geq 2$:

Theorem 5. *Let $\delta \geq 2$. Then*

(1) δ-gap-MSR-d *for any $d \geq 2$ is APX-hard. Moreover,* δ-gap-MSR-2*,* δ-gap-MSR-3*,* δ-gap-MSR-4*, and* δ-gap-MSR-d *are NP-hard to approximate within* 1.000431*,* 1.002114*,* 1.010661*, and $d/2^{O(\sqrt{\log d})}$, respectively, even if all markers are distinct and appear in positive orientation in each genomic map.*

(2) δ-gap-CMSR-d *for any $d \geq 2$ is APX-hard. Moreover,* δ-gap-CMSR-2*,* δ-gap-CMSR-3*,* δ-gap-CMSR-4*, and* δ-gap-CMSR-d *for any $d \geq 173$ are NP-hard to approximate within* 1.000625*,* 1.0101215*,* 1.0202429*, and $\frac{7}{6} - O(\log d/d)$, respectively, even if all markers are distinct and appear in positive orientation in each genomic map. If the number d of genomic maps is not a constant but is part of the input, then δ-gap-CMSR-d is NP-hard to approximate within any constant less than $10\sqrt{5} - 21 = 1.3606\ldots$, even if all markers are distinct and appear in positive orientation in each genomic map.*

We refer to arxiv.org/abs/0912.4935 for a complete manuscript combining the results in [13] and in this paper, and refer to [15,7] for some related results.

2 Preliminaries

L-reduction Given two optimization problems X and Y, an *L-reduction* [17] from X to Y consists of two polynomial-time functions f and g and two positive constants α and β satisfying the following two properties:

1. For every instance x of X, $f(x)$ is an instance of Y such that
$$\mathrm{opt}(f(x)) \leq \alpha \cdot \mathrm{opt}(x), \tag{1}$$
2. For every feasible solution y to $f(x)$, $g(y)$ is a feasible solution to x such that
$$|\mathrm{opt}(x) - \mathrm{val}(g(y))| \leq \beta \cdot |\mathrm{opt}(f(x)) - \mathrm{val}(y)|. \tag{2}$$

Here $\mathrm{opt}(x)$ denotes the value of the optimal solution to an instance x, and $\mathrm{val}(y)$ denotes the value of a solution y. The two properties of L-reduction imply the following inequality on the relative errors of approximation:

$$\frac{|\mathrm{opt}(x) - \mathrm{val}(g(y))|}{\mathrm{opt}(x)} \leq \alpha\beta \cdot \frac{|\mathrm{opt}(f(x)) - \mathrm{val}(y)|}{\mathrm{opt}(f(x))}.$$

A relative error of ϵ corresponds to an approximation factor of $1+\epsilon$ for a minimization problem, and corresponds to an approximation factor of $\frac{1}{1-\epsilon}$ for a maximization problem. Thus we have the following propositions:

1. For a minimization problem X and a minimization problem Y, if X is NP-hard to approximate within $1+\alpha\beta\epsilon$, then Y is NP-hard to approximate within $1+\epsilon$.
2. For a maximization problem X and a maximization problem Y, if X is NP-hard to approximate within $\frac{1}{1-\alpha\beta\epsilon}$, then Y is NP-hard to approximate within $\frac{1}{1-\epsilon}$.
3. For a maximization (resp. minimization) problem X and a minimization (resp. maximization) problem Y, if X is NP-hard to approximate within $\frac{1}{1-\alpha\beta\epsilon}$ (resp. $1+\alpha\beta\epsilon$), then Y is NP-hard to approximate within $1+\epsilon$ (resp. $\frac{1}{1-\epsilon}$).

APX-hard optimization problems. We review the complexities of some APX-hard optimization problems that will be used in our reductions.

- Max-IS-Δ is the problem Maximum Independent Set in graphs of maximum degree Δ. Max-IS-3 is APX-hard; see [4]. Moreover, Chlebík and Chlebíková [10] showed that Max-IS-3 and Max-IS-4 are NP-hard to approximate within 1.010661 and 1.0215517, respectively. Trevisan [18] showed that Max-IS-Δ is NP-hard to approximate within $\Delta/2^{O(\sqrt{\log \Delta})}$.
- Min-VC-Δ is the problem Minimum Vertex Cover in graphs of maximum degree Δ. Min-VC-3 is APX-hard; see [4]. Moreover, Chlebík and Chlebíková [10] showed that Min-VC-3 and Min-VC-4 are NP-hard to approximate within 1.0101215 and 1.0202429, respectively, and, for any $\Delta \geq 228$, Min-VC-Δ is NP-hard to approximate within $\frac{7}{6} - O(\log \Delta/\Delta)$. Dinur and Safra [11] showed that Minimum Vertex Cover is NP-hard to approximate within any constant less than $10\sqrt{5} - 21 = 1.3606\ldots$.

- Given a set X of n variables and a set $\mathcal{C}$ of m clauses, where each variable has exactly p literals (in p different clauses) and each clause is the disjunction of exactly q literals (of q different variables), Ep-Occ-Max-Eq-SAT is the problem of finding an assignment of X that satisfies the maximum number of clauses in $\mathcal{C}$. Note that $np = mq$. Berman and Karpinski [6] showed that E3-Occ-Max-E2-SAT is NP-hard to approximate within any constant less than $\frac{464}{463}$.
- Given d disjoint sets V_i of vertices, $1 \leq i \leq d$, and given a set $E \subseteq V_1 \times \cdots \times V_d$ of hyper-edges, d-Dimensional-Matching is the problem of finding a maximum-cardinality subset $M \subseteq E$ of pairwise-disjoint hyper-edges. Hazan, Safra, and Schwartz [12] showed that d-Dimensional-Matching is NP-hard to approximate within $\Omega(d/\log d)$.

Linear forest and linear arboricity. A *linear forest* is a graph in which every connected component is a path. The *linear arboricity* of a graph is the minimum number of linear forests into which the edges of the graph can be decomposed. Akiyama, Exoo, and Harary [2,3] conjectured that the linear arboricity of every graph G of maximum degree Δ satisfies $\mathrm{la}(G) \leq \lceil(\Delta+1)/2\rceil$. This conjecture has been confirmed for graphs of small constant degrees. In particular, the proof of the conjecture for $\Delta = 3$ and 4 are constructive [2,1,3] and lead to polynomial-time algorithms for decomposing any graph of maximum degree $\Delta = 3$ and 4 into at most $\lceil(\Delta+1)/2\rceil = 2$ and 3 linear forests, respectively. Also, the proof of the first upper bound on linear arboricity [3] implies a simple polynomial-time algorithm for decomposing any graph of maximum degree Δ into at most $\lceil 3\lceil\Delta/2\rceil/2\rceil$ linear forests.

Define

$$f(\Delta) = \max_G f(G),$$

where G ranges over all graphs of maximum degree Δ, and $f(G)$ denotes the number of linear forests that the algorithm in [3] decomposes G into. Then

$$\lceil(\Delta+1)/2\rceil \leq f(\Delta) \leq \lceil 3\lceil\Delta/2\rceil/2\rceil. \tag{3}$$

3 MSR-4 Is APX-Hard

In this section, we prove that MSR-4 is APX-hard by a simple L-reduction from Max-IS-3. Before we present the L-reduction, we first show that MSR-4 is NP-hard by a reduction in the classical style, which is perhaps more familiar to most readers. Throughout this paper, we follow this progressive format of presentation.

3.1 NP-Hardness Reduction from Max-IS-3 to MSR-4

Let G be a graph of maximum degree 3. Let n be the number of vertices in G. Partition the edges of G into two linear forests E_1 and E_2. Let V_1 and V_2 be the vertices of G that are *not* incident to any edges in E_1 and in E_2, respectively. We construct four genomic maps $G_{\rightarrow}$, $G_{\leftarrow}$, G_1, and G_2, where each map is a permutation of the following $2n$ distinct markers all in positive orientation:

– n pairs of vertex markers $\overset{i}{\subset}$ and $\overset{i}{\supset}$, $1 \leq i \leq n$.

$G_{\rightarrow}$ and $G_{\leftarrow}$ are concatenations of the n pairs of vertex markers with ascending and descending indices, respectively:

$$G_{\rightarrow} : \overset{1}{\subset}\overset{1}{\supset} \quad \cdots \quad \overset{n}{\subset}\overset{n}{\supset}$$
$$G_{\leftarrow} : \overset{n}{\subset}\overset{n}{\supset} \quad \cdots \quad \overset{1}{\subset}\overset{1}{\supset}$$

G_1 and G_2 are represented schematically as follows:

$$G_1 : \langle E_1 \rangle \ \langle V_1 \rangle$$
$$G_2 : \langle E_2 \rangle \ \langle V_2 \rangle$$

$\langle E_1 \rangle$ and $\langle E_2 \rangle$ consist of vertex markers of the vertices incident to the edges in E_1 and E_2, respectively. The markers of the vertices in each path $v_1 v_2 \dots v_k$ are grouped together in an interleaving pattern: for $1 \leq i \leq k$, the left marker of v_i, the right marker of v_{i-1} (if $i > 1$), the left marker of v_{i+1} (if $i < k$), and the right marker of v_i are consecutive.

$\langle V_1 \rangle$ and $\langle V_2 \rangle$ consist of vertex markers of the vertices in V_1 and V_2, respectively. The left marker and the right marker of each pair are consecutive.

This completes the construction. We refer to Figure 1 (a) and (b) for an example.

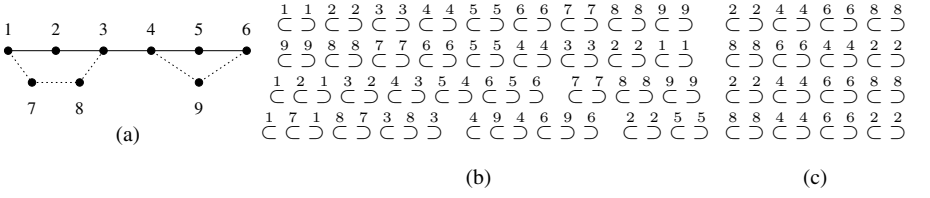


Fig. 1. (a) The graph G: E_1 is a single solid path $\langle 1, 2, 3, 4, 5, 6 \rangle$, E_2 consists of two dotted paths $\langle 1, 7, 8, 3 \rangle$ and $\langle 4, 9, 6 \rangle$, $V_1 = \{7, 8, 9\}$, $V_2 = \{2, 5\}$. (b) The four genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, G_2$. (c) The four subsequences of the genomic maps corresponding to the independent set $\{2, 4, 6, 8\}$ in the graph.

Two pairs of markers *intersect* in a genomic map if a marker of one pair appears between the two markers of the other pair. The following property of our construction is obvious:

Proposition 1. *Two vertices are adjacent in the graph G if and only if the corresponding two pairs of vertex markers intersect in one of the two genomic maps G_1, G_2.*

We say that four subsequences of the four genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, G_2$ are *canonical* if each strip of the subsequences is a pair of vertex markers. We have the following lemma on canonical subsequences:

Lemma 1. *In any four subsequences of the four genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, G_2$, respectively, each strip must be a pair of vertex markers.*

Proof. By construction, a strip cannot include two vertex markers of different indices because they appear in different orders in $G_{\rightarrow}$ and in $G_{\leftarrow}$. □

The following lemma establishes the NP-hardness of MSR-4:

Lemma 2. *The graph G has an independent set of at least k vertices if and only if the four genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, G_2$ have four subsequences whose total strip length l is at least $2k$.*

Proof. We first prove the "only if" direction. Suppose that the graph G has an independent set of at least k vertices. We will show that the four genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, G_2$ have four subsequences of total strip length at least $2k$. By Proposition 1, the k vertices in the independent set correspond to k pairs of vertex markers that do not intersect each other in the genomic maps. These k pairs of vertex markers induce a subsequence of length $2k$ in each genomic map. In each subsequence, the left marker and the right marker of each pair appear consecutively and compose a strip. Thus the total strip length is at least $2k$. We refer to Figure 1(c) for an example.

We next prove the "if" direction. Suppose that the four genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, G_2$ have four subsequences of total strip length at least $2k$. We will show that the graph G has an independent set of at least k vertices. By Lemma 1, each strip of the subsequences must be a pair of vertex markers. Thus we obtain at least k pairs of vertex markers that do not intersect each other in the genomic maps. Then, by Proposition 1, the corresponding set of at least k vertices in the graph G form an independent set. □

3.2 L-Reduction from Max-IS-3 to MSR-4

We present an L-reduction (f, g, α, β) from Max-IS-3 to MSR-4 as follows. The function f, given a graph G of maximum degree 3, constructs the four genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, G_2$ as in the NP-hardness reduction. Let k^* be the number of vertices in a maximum independent set in G, and let l^* be the maximum total strip length of any four subsequences of $G_{\rightarrow}, G_{\leftarrow}, G_1, G_2$, respectively. By Lemma 2, we have

$$l^* = 2k^*.$$

Choose $\alpha = 2$, then property (1) of L-reduction is satisfied.

The function g, given four subsequences of the four genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, G_2$, respectively, returns an independent set of vertices in the graph G corresponding to the pairs of vertex markers that are strips of the subsequences. Let l be the total strip length of the subsequences, and let k be the number of vertices in the independent set returned by the function g. Then $k \geq l/2$. It follows that

$$|k^* - k| = k^* - k \leq l^*/2 - l/2 = |l^* - l|/2.$$

Choose $\beta = 1/2$, then property (2) of L-reduction is also satisfied.

We have obtained an L-reduction from Max-IS-3 to MSR-4 with $\alpha\beta = 1$. Chlebík and Chlebíková [10] showed that Max-IS-3 is NP-hard to approximate within 1.010661. It follows that MSR-4 is also NP-hard to approximate within 1.010661. The lower bound extends to MSR-d for all constants $d \geq 4$.

The L-reduction from Max-IS-3 to MSR-4 can be obviously generalized:

Lemma 3. *Let $\Delta \geq 3$ and $d \geq 4$. If there is a polynomial-time algorithm for decomposing any graph of maximum degree Δ into $d-2$ linear forests, then there is an L-reduction from* Max-IS-Δ *to* MSR-d *with constants $\alpha = 2$ and $\beta = 1/2$.*

4 An Asymptotic Lower Bound for MSR-d

In this section, we derive an asymptotic lower bound for approximating MSR-d by an L-reduction from d-Dimensional-Matching to MSR-$(d+2)$.

4.1 NP-Hardness Reduction from d-Dimensional-Matching to MSR-$(d+2)$

Let $E \subseteq V_1 \times \cdots \times V_d$ be a set of n hyper-edges over d disjoint sets V_i of vertices, $1 \leq i \leq d$. We construct two genomic maps $G_{\rightarrow}$ and $G_{\leftarrow}$, and d genomic maps G_i, $1 \leq i \leq d$, where each map is a permutation of the following $2n$ distinct markers all in positive orientation:

- n pairs of edge markers $\overset{i}{\subset}$ and $\overset{i}{\supset}$, $1 \leq i \leq n$.

The two genomic maps $G_{\rightarrow}$ and $G_{\leftarrow}$ are concatenations of the n pairs of edge markers with ascending and descending indices, respectively:

$$G_{\rightarrow} : \overset{1}{\subset}\ \overset{1}{\supset} \quad \cdots \quad \overset{n}{\subset}\ \overset{n}{\supset}$$
$$G_{\leftarrow} : \overset{n}{\subset}\ \overset{n}{\supset} \quad \cdots \quad \overset{1}{\subset}\ \overset{1}{\supset}$$

Each genomic map G_i corresponds to a vertex set $V_i = \{v_{i,j} \mid 1 \leq j \leq |V_i|\}$, $1 \leq i \leq d$, and is represented schematically as follows:

$$G_i : \quad \cdots \quad \langle v_{i,j} \rangle \quad \cdots$$

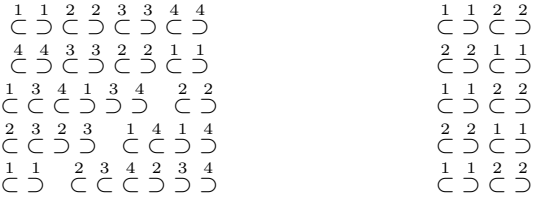
Here each $\langle v_{i,j} \rangle$ consists of the edge markers of hyper-edges containing the vertex $v_{i,j}$, grouped together such that the left markers appear with ascending indices before the right markers also with ascending indices. This completes the construction. We refer to Figure 2(a) for an example.

The following property of our construction is obvious:

Proposition 2. *Two hyper-edges in E intersect if and only if the corresponding two pairs of edge markers intersect in one of the d genomic maps G_i, $1 \leq i \leq d$.*

The following lemma is analogous to Lemma 1:

Lemma 4. *In any $d+2$ subsequences of the $d+2$ genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, \ldots, G_d$, respectively, each strip must be a pair of edge markers.*



(a) (b)

Fig. 2. MSR-5 construction for the 3-Dimensional-Matching instance $V_1 = \{v_{1,1}, v_{1,2}\}$, $V_2 = \{v_{2,1}, v_{2,2}\}$, $V_3 = \{v_{3,1}, v_{3,2}\}$, and $E = \{\, e_1 = (v_{1,1}, v_{2,2}, v_{3,1}),\ e_2 = (v_{1,2}, v_{2,1}, v_{3,2}),\ e_3 = (v_{1,1}, v_{2,1}, v_{3,2}),\ e_4 = (v_{1,1}, v_{2,2}, v_{3,2}) \,\}$. (a) The five genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, G_2, G_3$. (b) The five subsequences of the genomic maps corresponding to the subset $\{e_1, e_2\}$ of pairwise-disjoint hyper-edges.

Analogous to Lemma 2, the following lemma establishes the NP-hardness of MSR-d:

Lemma 5. *The set E has a subset of k pairwise-disjoint hyper-edges if and only if the $d+2$ genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, \ldots, G_d$ have $d+2$ subsequences whose total strip length l is at least $2k$.*

Proof. We first prove the "only if" direction. Suppose that the set E has a subset of at least k pairwise-disjoint hyper-edges. We will show that the $d+2$ genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, \ldots, G_d$ have $d+2$ subsequences of total strip length at least $2k$. By Proposition 2, the k pairwise-disjoint hyper-edges correspond to k pairs of edge markers that do not intersect each other in the genomic maps. These k pairs of edge markers induce a subsequence of length $2k$ in each genomic map. In each subsequence, the left marker and the right marker of each pair appear consecutively and compose a strip. Thus the total strip length is at least $2k$. We refer to Figure 2(b) for an example.

We next prove the "if" direction. Suppose that the $d+2$ genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, \ldots, G_d$ have $d+2$ subsequences of total strip length at least $2k$. We will show that the set E has a subset of at least k pairwise-disjoint hyper-edges. By Lemma 1, each strip of the subsequences must be a pair of edge markers. Thus we obtain at least k pairs of edge markers that do not intersect each other in the genomic maps. Then, by Proposition 2, the corresponding set of at least k hyper-edges in E are pairwise-disjoint. □

4.2 L-Reduction from d-Dimensional-Matching to MSR-$(d+2)$

We present an L-reduction (f, g, α, β) from d-Dimensional-Matching to MSR-$(d+2)$ as follows. The function f, given a set $E \subseteq V_1 \times \cdots \times V_d$ of hyper-edges, constructs the $d+2$ genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, \ldots, G_d$ as in the NP-hardness reduction. Let k^* be the maximum number of pairwise-disjoint hyper-edges in E, and let l^* be the maximum total strip length of any $d+2$ subsequences of $G_{\rightarrow}, G_{\leftarrow}, G_1, \ldots, G_d$, respectively. By Lemma 5, we have

$$l^* = 2k^*.$$

Choose $\alpha = 2$, then property (1) of L-reduction is satisfied.

The function g, given $d + 2$ subsequences of the $d + 2$ genomic maps $G_{\rightarrow}, G_{\leftarrow}, G_1, \ldots, G_d$, respectively, returns a subset of pairwise-disjoint hyper-edges in E corresponding to the pairs of edge markers that are strips of the subsequences. Let l be the total strip length of the subsequences, and let k be the number of pairwise-disjoint hyper-edges returned by the function g. Then $k \geq l/2$. It follows that

$$|k^* - k| = k^* - k \leq l^*/2 - l/2 = |l^* - l|/2.$$

Choose $\beta = 1/2$, then property (2) of L-reduction is also satisfied.

We have obtained an L-reduction from d-Dimensional-Matching to MSR-$(d + 2)$ with $\alpha\beta = 1$. Hazan, Safra, and Schwartz [12] showed that d-Dimensional-Matching is NP-hard to approximate within $\Omega(d/\log d)$. It follows that MSR-d is also NP-hard to approximate within $\Omega(d/\log d)$. This completes the proof of Theorem 2.

5 A Polynomial-Time $2d$-Approximation for MSR-d

In this section we prove Theorem 3. We briefly review the two previous algorithms [20, 9] for this problem. The first algorithm for MSR-2 is a simple heuristic due to Zheng, Zhu, and Sankoff [20]:

1. Extract a set of pre-strips from the two genomic maps;
2. Compute an independent set of strips from the pre-strips.

This algorithm is inefficient because the number of pre-strips could be exponential in the sequence length, and furthermore the problem Maximum-Weight Independent Set in general graphs is NP-hard.

Chen, Fu, Jiang, and Zhu [9] presented a $2d$-approximation algorithm for MSR-d. For any $d \geq 2$, a *d-interval* is the union of d disjoint intervals in the real line, and a *d-interval graph* is the intersection graph of a set of d-intervals, with a vertex for each d-interval, and with an edge between two vertices if and only the corresponding d-intervals overlap. The $2d$-approximation algorithm [9] works as follows:

1. Compose a set of d-intervals, one for each combination of d substrings of the d genomic maps, respectively. Assign each d-interval a weight equal to the length of a longest common subsequence (which may be reversed and negated) in the corresponding d substrings.
2. Compute a $2d$-approximation for Maximum-Weight Independent Set in the resulting d-interval graph using Bar-Yehuda et al.'s fractional local-ratio algorithm [5].

Let n be the number of markers in each genomic map. Then the number of d-intervals composed by this algorithm is $\Theta(n^{2d})$ because each of the d genomic maps has $\Theta(n^2)$ substrings. Consequently the running time of this algorithm can be exponential if the number d of genomic maps is not a constant but is part of the input. In the following, we show that if all markers are distinct in each genomic map (as discussed earlier, this is a reasonable assumption in application), then the running time of the $2d$-approximation algorithm can be improved to polynomial for all $d \geq 2$. This improvement is achieved by composing a smaller set of candidate d-intervals in step 1 of the algorithm.

The idea is actually quite simple and has been used many times previously [16,14,8]. Note that any strip of length $l > 3$ is a concatenation of shorter strips of lengths 2 and 3, for example, $4 = 2 + 2$, $5 = 2 + 3$, etc. Since the objective is to maximize the total strip length, it suffices to consider only short strips of lengths 2 and 3 in the genomic maps, and to enumerate only candidate d-intervals that correspond to these strips. When each genomic map is a signed permutation of the same n distinct markers, there are at most $\binom{n}{2} + \binom{n}{3} = O(n^3)$ strips of lengths 2 and 3, and for each strip there is a unique shortest substring of each genomic map that contains all markers in the strip. Thus we compose only $O(n^3)$ d-intervals, and improve the running time of the $2d$-approximation algorithm to polynomial for all $d \geq 2$. This completes the proof of Theorem 3.

6 Inapproximability Results for Related Problems

In this section we prove Theorem 4 and Theorem 5.

For any d, the decision problems of MSR-d and CMSR-d are equivalent. Thus the NP-hardness of MSR-d implies the NP-hardness of CMSR-d, although the APX-hardness of MSR-d does not necessarily imply the APX-hardness of CMSR-d. Note that the two problems Max-IS-Δ and Min-VC-Δ complement each other just as the two problems MSR-d and CMSR-d complement each other. Thus our NP-hardness reduction from Max-IS-3 to MSR-3 in [13] can be immediately turned into an NP-hardness reduction from Min-VC-3 to CMSR-3. Similarly, our L-reduction from Max-IS-3 to MSR-3 in [13] can be adapted into an L-reduction from Min-VC-3 to CMSR-3 with $\alpha = 2$ and $\beta = 1/2$, and our L-reduction from E3-Occ-Max-E2-SAT to MSR-2 in [13] can be adapted into an L-reduction from E3-Occ-Max-E2-SAT to CMSR-2 with $\alpha = 62/9$ and $\beta = 1/2$. An asymptotic lower bound for CMSR-d and a lower bound for CMSR-d with unbounded d can also be obtained. This completes the proof of Theorem 4.

It is easy to check that all instances of MSR-d and CMSR-d in our constructions for Theorem 2 and Theorem 4 admit optimal solutions in canonical form with maximum gap 2, except for the following two cases:

1. In the L-reduction from Ep-Occ-Max-Eq-SAT to MSR-2 and CMSR-2, a strip that is a pair of literal markers has a gap of $q - 1$, which is larger than 2 for $q \geq 4$.
2. In the L-reduction from d-Dimensional-Matching to MSR-$(d + 2)$, a strip that is a pair of edge markers may have an arbitrarily large gap if it corresponds to one of many hyper-edges that share a single vertex.

To extend our results in Theorem 2 and Theorem 4 to the corresponding results in Theorem 5, the first case does not matter because we set the parameter q to 2 when deriving the lower bounds for MSR-2 and CMSR-2 from the lower bound for E3-Occ-Max-E2-SAT. The second case is more problematic, and we have to use a different L-reduction to obtain a slightly weaker asymptotic lower bound for δ-gap-MSR-d. Trevisan [18] showed that Max-IS-Δ is NP-hard to approximate within $\Delta/2^{O(\sqrt{\log \Delta})}$. By Lemma 3, there is an L-reduction from Max-IS-Δ to δ-gap-MSR-$(f(\Delta) + 2)$ with $\alpha\beta = 1$. By the two inequalities in (3), we have $f(\Delta) + 2 = \Theta(\Delta)$. Thus δ-gap-MSR-d is NP-hard to approximate within $d/2^{O(\sqrt{\log d})}$. This completes the proof of Theorem 5.

References

1. Akiyama, J., Chvátal, V.: A short proof of the linear arboricity for cubic graphs. Bull. Liber. Arts & Sci. NMS 2, 1–3 (1981)
2. Akiyama, J., Exoo, G., Harary, F.: Covering and packing in graphs III: cyclic and acyclic invariants. Mathematica Slovaca 30, 405–417 (1980)
3. Akiyama, J., Exoo, G., Harary, F.: Covering and packing in graphs IV: linear arboricity. Networks 11, 69–72 (1981)
4. Alimonti, P., Kann, V.: Some APX-completeness results for cubic graphs. Theoretical Computer Science 237, 123–134 (2000)
5. Bar-Yehuda, R., Halldórsson, M.M., Naor, J.(S.), Shachnai, H., Shapira, I.: Scheduling split intervals. SIAM Journal on Computing 36, 1–15 (2006)
6. Berman, P., Karpinski, M.: Improved approximation lower bounds on small occurrence optimization. Electronic Colloquium on Computational Complexity, Report TR03-008 (2003)
7. Bulteau, L., Fertin, G., Jiang, M., Rusu, I.: Tractablity and approximability of maximal strip recovery (submitted)
8. Bulteau, L., Fertin, G., Rusu, I.: Maximal strip recovery problem with gaps: hardness and approximation algorithms. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 710–719. Springer, Heidelberg (2009)
9. Chen, Z., Fu, B., Jiang, M., Zhu, B.: On recovering syntenic blocks from comparative maps. Journal of Combinatorial Optimization 18, 307–318 (2009)
10. Chlebík, M., Chlebíková, J.: Complexity of approximating bounded variants of optimization problems. Theoretical Computer Science 354, 320–338 (2006)
11. Dinur, I., Safra, S.: On the hardness of approximating minimum vertex cover. Annals of Mathematics 162, 439–485 (2005)
12. Hazan, E., Safra, S., Schwartz, O.: On the complexity of approximating k-set packing. Computational Complexity 15, 20–39 (2006)
13. Jiang, M.: Inapproximability of maximal strip recovery. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 616–625. Springer, Heidelberg (2009)
14. Jiang, M.: Approximation algorithms for predicting RNA secondary structures with arbitrary pseudoknots. IEEE/ACM Transactions on Computational Biology and Bioinformatics 7, 323–332 (2010)
15. Jiang, M.: On the parameterized complexity of some optimization problems related to multiple-interval graphs. In: Javed, A. (ed.) CPM 2010. LNCS, vol. 6129, pp. 125–137. Springer, Heidelberg (2010)
16. Lyngsø, R.B.: Complexity of pseudoknot prediction in simple models. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 919–931. Springer, Heidelberg (2004)
17. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. Journal of Computer and System Sciences 43, 425–440 (1991)
18. Trevisan, L.: Non-approximability results for optimization problems on bounded degree instances. In: Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC 2001), pp. 453–461 (2001)
19. Wang, L., Zhu, B.: On the tractability of maximal strip recovery. In: TAMC 2009. LNCS, vol. 5532, pp. 400–409. Springer, Heidelberg (2009)
20. Zheng, C., Zhu, Q., Sankoff, D.: Removing noise and ambiguities from comparative maps in rearrangement analysis. IEEE/ACM Transactions on Computational Biology and Bioinformatics 4, 515–522 (2007)

Minimizing Total Variation for Field Splitting with Feathering in Intensity-Modulated Radiation Therapy*

Yunlong Liu[1] and Xiaodong Wu[1,2]

[1] Electrical and Computer Engineering, The University of Iowa
[2] Department of Radiation Oncology, The University of Iowa
{yunlong-liu,xiaodong-wu}@uiowa.edu

Abstract. In this paper, we study an interesting geometric partition problem, called optimal field splitting, which arises in Intensity-Modulated Radiation Therapy (IMRT). In current clinical practice, a multileaf collimator (MLC) with a maximum leaf spread constraint is used to deliver the prescribed radiation intensity maps (IMs). However, the maximum leaf spread of an MLC may require to split a large IM into several overlapping sub-IMs with each being delivered separately. We develop an efficient algorithm for solving the field splitting problem while minimizing the total variation of the resulting sub-IMs, thus improving the treatment delivery efficiency. Our basic idea is to formulate the field splitting problem as computing a shortest path in a directed acyclic graph, which expresses a special "layered" structure. The edge weights in the graph can be computed by solving an optimal vector decomposition problem using local searching and the proximity scaling technique as we can prove the $L^\natural$-convexity and totally unimodularity of the problem. Moreover, the edge weights of the graph satisfy the Monge property, which enables us to solve this shortest path problem by examining only a small portion of the graph, yielding a time-efficient algorithm.

1 Introduction

In this paper, we study an interesting geometric optimization problem, called *optimal field splitting with feathering*, which arises in a modern cancer therapy technique named *Intensity-Modulated Radiation Therapy* (IMRT). The quality of IMRT depends on the ability to accurately and efficiently deliver the prescribed dose distributions of radiation, commonly called *intensity maps* (IMs), to the tumor while sparing the surrounding normal tissues. An IM is specified by a set of nonnegative integers on a 2-D grid, each of which indicates the amount of radiation to be delivered to the corresponding body region.

In current clinical practice, an advanced tool for IM delivery is the multileaf collimator (MLC) [17]. An MLC consists of several pairs of uniform-sized rectangular tungsten alloy leaves which can move left and right to form different

* This research was supported in part by the NSF grants CCF-0830402 and CCF-0844765, and the NIH grant K25-CA123112.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 65–76, 2010.

rectilinear regions, called *MLC-apertures*. The cross-section of a radiation beam is shaped by an MLC-aperture. Due to the mechanical design of the MLCs, there are some restrictions on the beam-shaping regions [17]. One common constraint is the *maximum leaf spread*: each MLC leaf cannot travel away from the vertical center line of the MLC more than a certain distance Δ (e.g., $\Delta = 14.5$ cm for the Varian MLCs). Thus, an IM with a large width may not be enclosed within a single MLC-aperture. Today large intensity maps frequently occur [18]. Such a large IM needs to be split into multiple sub-IMs, each being delivered separately.

One commonly used IMRT technique for IM delivery using an MLC is the "step-and-shoot" method [4,17].Mathematically speaking, the "step-and-shoot" technique can be viewed as an IM segmentation problem: given an intensity map $A = (a_{i,j})_{m\times n}$, decompose A into $A = \sum_{k=1}^{\kappa} \alpha_k S_k$, where S_k is a special 0-1 matrix specifying a segment that can be conformed by an MLC-aperture, α_k is the amount of radiation delivered through S_k, and κ is the number of segments used to deliver A. The MLC leaves move to form each of those κ segments, and to deliver α_i units of radiation for the corresponding segment to achieve the delivery of the whole IM. The number of monitor units (MUs), which is determined by $\sum_{k=1}^{\kappa} \alpha_k$, is a critical measure for the efficiency of the step-and-shoot delivery. In fact, it determines the radiation delivery time which is ideally minimized to reduce the impact of intra-fraction organ motions [17]. In addition, the reduction of delivery time enables to minimize the loss of biological effectiveness, improving the effectiveness of IMRT [6]. In general, the minimum number of MUs for delivering an IM is closely related to the complexity of the IM, which is unfortunately not well defined. Some researchers use the sum of positive gradients to measure the complexity [20,2,3]. In this paper, we adopt the *total variation* of an IM to measure the complexity [16]. More precisely, the complexity of an IM A, $C(A) = \sum_{i=1}^{m} \left(a_{i,1}^2 + \sum_{j=2}^{n-1}(a_{i,j} - a_{i,j+1})^2 + a_{i,n}^2\right)$. Süss and Küfer found that if the total variation of an intensity map increases, the minimum number of MUs used to deliver it is expected to increase as well [18]. Thus, the total variation is a good measure of the complexity of an IM.

The splitting of an IM may result in a prolonged delivery time, thus degrading the treatment quality. The goal of the *optimal field splitting*, is to find a way to split a large IM into a set of sub-IMs whose sizes are no larger than a threshold, such that the total complexity of those sub-IMs is minimized, yielding optimized treatment quality.

A natural way for splitting a large IM is to use vertical straight lines, yielding abutting sub-IMs. However, if the borders of two abutting sub-IMs are not precisely aligned, hotspots or coldspots will be introduced. That kind of field mismatching problem is common due to the uncertainties of the patient setup and organ motion [8,18]. To alleviate such a problem, a commonly used medical practice is a so-called *field feathering* technique [8,18]. With this technique, a large IM is split into a set of sub-IMs, such that each sub-IM S_k is subject to the maximum leaf spread constraint, and any two adjacent sub-IMs overlap over a central *feathering region*. Each cell in the feathering region belongs to two adjacent sub-IMs, with non-negative intensity value in both sub-IMs.

Recently, several field splitting algorithms have been reported in the literatures on improving the delivery efficiency and accuracy for large IMs. To our best knowledge, Kamath *et al.* first gave an $\mathcal{O}(mn^2)$ time algorithm to split along column using vertical lines into *at most three* sub-IMs while minimizing the total MUs, with or without feathering [9]. They also gave an algorithm for a more general field splitting with the field width as the only constraint. However, the algorithm again only works for no more than three sub-IMs [8]. In practice, the current use of the high resolution motorized micro multileaf collimator, e.g. one manufactured by MRC systems GmbH Heidelberg which has a maximum field width of 7.2 cm [15], may require to split an IM into more than three sub-IMs to treat large tumor sites. Wu formulated the field splitting without feathering problem for an arbitrary field width using vertical lines as a k-link shortest path problem and developed an $\mathcal{O}(mn\varpi)$ time algorithm, where ϖ is the maximum allowed field width [19]. Chen *et al.* developed a field splitting algorithm in which the total MUs is minimized for an arbitrarily large IM field. Their algorithm runs in time $\mathcal{O}(mn + m\xi^{d-2})$, where d is the number of the resulting sub-IMs and ξ is the reminder of n divided by ϖ [5]. By exploring the properties of the complexity function that adopts the sum of positive gradients of an IM, Wu *et al.* achieved a field splitting algorithm that runs in time $\mathcal{O}(mn\alpha(\varpi))$, where $\alpha(\cdot)$ is the inverse Ackermann function [20].

In this paper, we study the following *optimal field splitting with minimized total variation (OFS-TV)* problem. Given an IM $A = (a_{i,j})_{m\times n}$ of size $m \times n$, an integral maximum field width $\varpi > 0$, and the feathering region width range $0 < \delta < \Delta < \varpi$, split A with vertical lines into a sequence of $d = \lceil \frac{n-\delta}{\varpi-\delta} \rceil \geq 2$ sub-IMs, such that: (1) the width of each sub-IM, except the last one, is ϖ, and the width of the last one is larger than 0 and no larger than ϖ; (2) any two neighboring sub-IMs in the sequence overlap each other and the width of overlapping (feathering) region is between δ and Δ; (3) no sub-IM overlaps completely with its neighbors; and (4) the total complexity of all these sub-IMs is minimized.

By exploring convexity of the OFS-TV problem, we develop an efficient $\mathcal{O}(mn\varpi^2\alpha(\varpi)\log U)$ time algorithm, where U is the largest entry in the input IM. The algorithmic techniques used include local searching, proximity scaling, and Monge matrix searching. Although we adopt a similar algorithm framework as in Wu *et al.* [20], judicious characterization of the total variation measure of an IM is critical for achieving our efficient algorithm. In our algorithm, we first model the computation of an "optimal" set of (d-1) feathering regions as a shortest path problem in a directed acyclic graph (DAG). This DAG has a special "layered" structure, which consists of d layers of nodes with any two adjacent layers inducing a bipartite graph. We are able to calculate each edge weight in $\mathcal{O}(\varpi^3 \log U)$ time using the local searching and proximity scaling techniques. Moreover, the edge weights of the DAG satisfy the Monge property [1]. Thus, we can solve this shortest path problem by examining only a small portion of the graph, and our algorithm runs in an $\mathcal{O}(mn\varpi^2\alpha(\varpi)\log U)$ time. Then, the optimal set of (d-1) feathering regions are decomposed to yield an optimal split.

In practice, if U is not much larger than ϖ, we can obtain an $\mathcal{O}(mnU\alpha(\varpi))$ time algorithm without using the proximity scaling technique.

2 The Algorithm

In this section, we present our $\mathcal{O}(mn\varpi^2\alpha(\varpi)\log U)$ time algorithm for solving the optimal field splitting with minimized total variation (OFS-TV) problem. The algorithm consists of two major steps: (1) Computing an optimal set of $(d-1)$ feathering regions; and (2) decomposing each of those feathering regions to form an optimal split consisting of d sub-IMs.

2.1 The Shortest Path Model for Computing an Optimal Set of Feathering Regions

Denote the j-th column of IM A as $A[j]$, and $A[j..k]$ is a sub-matrix consists of all elements of A from column j to column k. Since the widths of the first $d-1$ sub-IM are fixed as ϖ (to make full use of the field width), only d vertical lines $\{j_1, j_2, \ldots, j_d\}$ are needed to identify the starting column of each sub-IM (j_1 is fixed as 1). Then the k-th feathering region can be easily determined as $F_k = A[j_{k+1}..j_k+\varpi-1]$. And F_k is somehow decomposed into $F_k^{(0)}$ and $F_k^{(1)}$ such that $F_k = F_k^{(0)} + F_k^{(1)}$. A feasible *split* of IM A, denote by $\mathcal{S} = \{S_1, S_2, \ldots, S_d\}$, is defined as follows. For each $k \in \{1, 2, \ldots, d\}$, $S_k = F_{k-1}^{(1)} || A[j_{k-1} + \varpi..j_{k+1} - 1] || F_k^{(0)}$, where $||$ is a concatenation operator, $F_0^{(1)} = F_d^{(0)} = \emptyset$, $j_0 = -\varpi + 1$ and $j_{d+1} = n + 1$. The decomposition of each feathering region F_k may increase the total complexity. Our goal is to find a set $\mathcal{F}$ of the $(d-1)$ feathering regions such that the total increase of the complexity introduced by the decomposition is minimized.

In this paper, we adopt a similar shortest path model as used in Wu *et al.* [20] to compute an optimal set of $(d-1)$ feathering regions. A weighted directed acyclic graph (DAG) $G = (V, E)$ is constructed, as follows.

- For each column j of A, there is exactly a node $u_j \in V$. The nodes are organized into d layers, with each layer L_k containing all the possible starting columns of the sub-IM S_k. As shown in Wu *et al.* [20], the first layer L_1 consists of only one node u_1, i.e., the first sub-IM S_1 has to start at the first column of A; each other layer L_k $(k = 2, 3, \ldots, d)$ contains $\mu = (n - \delta) \bmod (\varpi - \delta)$ nodes $\{u_j \mid (k-1)(\varpi-\delta)+1-\mu \le j \le (k-1)(\varpi-\delta)+1\}$; and the layers are mutually exclusive (i.e., $L_k \cap L_{k+1} = \emptyset$ for $k = 1, 2, \ldots, d-1$).
- For any node $u_i \in L_k$ and $u_j \in L_{k+1}$, there is edge (u_i, u_j) if and only if the width of the overlap region of the two corresponding sub-IMs is between δ and Δ. The edge weight $w(u_i, u_j)$ is computed as the minimum increased complexity Δ_{cpl} of the corresponding feathering region $A[j \,..\, i + \varpi - 1]$.
- A dummy sink node t is added, and each node in L_d has a directed edge to t with a weight of 0.

It is clear that a shortest u_1-to-t path, encoding all the starting columns of the sub-IMs, thus defines an optimal set of $(d-1)$ feathering regions. Given that the weight of each edge can be accessed in $\mathcal{O}(1)$ time, it is well-known that computing a shortest path in G takes $\mathcal{O}(|V|+|E|)$ time, where $|V| = n+1$ and $|E| = \mathcal{O}(n\varpi)$. Unfortunately, we are not able to compute the weight of each edge in $\mathcal{O}(1)$ time. The edge weight computation becomes the bottleneck for solving the OSF-TV problem. Next, we show how to compute the edge weights efficiently and how to use the Monge property to avoid the computation of non-necessary edge weights to speed up the shortest path computation in G.

2.2 Efficient Algorithm for the Edge Weight Computation

Note that an edge $(u_i, u_j) \in E$ with $u_i \in L_k$ and $u_j \in L_{k+1}$ defines a feathering region $F_k = A[j \,..\, i+\varpi-1]$ and the weight of this edge equals to the minimum increase of the complexity of F_k. In this section, by exploiting the convexity of the problem, we develop our efficient algorithm for computing the weight of each edge $(u_i, u_j) \in E$ based on the local searching and proximity scaling techniques.

Consider the feathering region $F_k = A[l..r]$ with a width of $\omega = r - l + 1$, which is the overlapping region of S_k and S_{k+1}. Assume that F_k is decomposed into $F_k^{(0)} = (x_{i,j})_{m\times\omega}$ and $F_k^{(1)} = (y_{i,j})_{m\times\omega}$. Then the contribution $R(F_k)$ of F_k to the complexity of IM A is $\sum_{i=1}^{m}\sum_{j=l}^{r+1}(a_{i,j}-a_{i,j-1})^2$. While the contribution $R(F_k^{(0)})$ to the complexity of S_k is

$$R(F_k^{(0)}) = \sum_{i=1}^{m}\left((x_{i,l}-a_{i,l-1})^2 + \sum_{j=l+1}^{r}(x_{i,j}-x_{i,j-1})^2 + x_{i,r}^2\right) \tag{1}$$

and similarly, the contribution $R(F_k^{(1)})$ to the complexity of S_{k+1} is

$$R(F_k^{(1)}) = \sum_{i=1}^{m}\left(y_{i,l}^2 + \sum_{j=l+1}^{r}(y_{i,j}-y_{i,j-1})^2 + (a_{i,r+1}-y_{i,r})^2\right) \tag{2}$$

Therefore, it is natural to define the *increase of the complexity* due to the decomposition of the feathering region as $R(F_k^{(0)}) + R(F_k^{(1)}) - R(F_k)$.

Note that the term $R(F_k)$ is a constant for any decomposition of F_k, and the decomposition of F_k can be performed on each row independently. We define the following *optimal vector decomposition* (OVD) problem. Given a non-negative integer vector $\mathbf{b} = (b_1, b_2, \ldots, b_N)$, decompose $\mathbf{b}$ into two non-negative integer vectors $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_N)$, such that (1) $x_1 = b_1$ and $y_N = b_N$; (2) $\forall j \in [1, N]$, $b_j = x_j + y_j$; and (3) $W_{ovd}(\mathbf{x}) + W_{ovd}(\mathbf{y})$ is minimized, where the function $W_{ovd}(\mathbf{z}) = z_1^2 + \sum_{j=2}^{N}(z_j - z_{j-1})^2 + z_N^2$ for a given vector $\mathbf{z} = (z_1, z_2, \ldots, z_N)$.

Then, for a given feathering region $F_k = A[l..r]$, we can apply the OVD algorithm to each extended row of F_k, i.e. $(a_{i,l-1}, a_{i,l}, \ldots, a_{i,r}, a_{i,r+1})$ ($a_{i,0} = a_{i,n+1} = 0$), yielding an optimal decomposition $\mathbf{x}_i$ and $\mathbf{y}_i$ with $W_{ovd}(\mathbf{x}_i) + W_{ovd}(\mathbf{y}_i)$ minimized. Clearly, the optimal decomposition of F_k is specified by $F_k^{(0)} = (\mathbf{x}_i)_{i=1}^m$ and $F_k^{(1)} = (\mathbf{y}_i)_{i=1}^m$. Thus, $F_k^{(0)}$ and $F_k^{(1)}$ is an optimal decomposition of F_k minimizing the increase of the complexity.

Unfortunately, the linear algorithm developed in [20,5] for the OVD problem, in which a different IM complexity measure was used, is not applicable for our case. A new technique is needed.

The graph model for the OVD problem

Given a non-negative integer vector $\mathbf{b} = (b_1, b_2, \ldots, b_N)$, we define an edge-weighted DAG $H = (V_H, E_H)$ for the OVD problem, as follows.

The element b_1 (resp., b_N) of $\mathbf{b}$ corresponds to exactly one node $v_1(b_1)$ (resp., $v_N(0)$) in V_H, briefly called the *source* (resp., *sink*) node s (resp., t) of H. For every other element b_j ($j = 2, 3, \ldots, N-1$), there is a set $Col(j)$ of $b_j + 1$ nodes in H corresponding to b_j, with $Col(j) = \{v_j(h) \mid h = 0, 1, \ldots, b_j\}$, namely the b_j-*column* of H. Intuitively, the nodes in $Col(j)$ give all possible distinct ways to decompose b_j into two non-negative integers (i.e., each node $v_j(h)$ corresponds to decomposing b_j into h and $b_j - h$). Note that b_1-column (resp., b_N-column) consists of only one node $v_1(b_1)$ (resp., $v_N(0)$). For any two adjacent columns $Col(j)$ and $Col(j+1)$ ($j = 1, 2, \ldots, N-1$), each node $v_j(h) \in Col(j)$ has a directed edge e to every node $v_{j+1}(h')$, with an edge *weight* $w(e) = (h' - h)^2 + ((b_{j+1} - h') - (b_j - h))^2$.

Consider any s-to-t path p in H, with $p = v_1(h_1) \rightarrow v_2(h_2) \rightarrow \ldots \rightarrow v_{N-1}(h_{N-1}) \rightarrow v_N(h_N)$, where $h_1 = b_1$ and $h_N = 0$ (i.e., $v_1(h_1)$ is the source s and $v_N(h_N)$ is the sink t). Let $\mathbf{x}(p) = (x_1, x_2, \ldots, x_N)$ and $\mathbf{y}(p) = (y_1, y_2, \ldots, y_N)$ be two non-negative integer vectors defined from the path p, in the following way: for each $j = 1, 2, \ldots, N$, $x_j = h_j$ and $y_j = b_j - h_j$. Note that each s-to-t path p in H actually define a feasible decomposition of $\mathbf{b}$, i.e., $\mathbf{b} = \mathbf{x}(p) + \mathbf{y}(p)$. The total weight of $\mathbf{x}(p)$ and $\mathbf{y}(p)$, $W_{ovd}(\mathbf{x}(p)) + W_{ovd}(\mathbf{y}(p))$, equals to the total sum of the weights of the edges on p, i.e., $w(p) = W_{ovd}(\mathbf{x}(p)) + W_{ovd}(\mathbf{y}(p))$. Hence, a shortest s-to-t path in H, which can be computed in $\mathcal{O}(NU^2)$ time ($U = \max_{j=1}^N b_j$), specifies an optimal decomposition of $\mathbf{b}$.

This is a pseudo-polynomial time algorithm for the OVD problem, which may not be efficient enough, especially when the elements of $\mathbf{b}$ are large. We next exploit the convexity and unimodularity of the problem to achieve our polynomial $\mathcal{O}(N^3 \log U)$ time OVD algorithm.

Local Searching

Assume that $(\mathbf{x}, \mathbf{y})$ is a decomposition of the vector $\mathbf{b}$. In fact, $\mathbf{x}$ defines an s-to-t path in graph H and vice verse. Let $\Delta x_j = x_j - x_{j-1}$. Then, the objective function of the OVD problem $\mathcal{E}(\mathbf{x}) = W_{ovd}(\mathbf{x}) + W_{ovd}(\mathbf{y}) = \sum_{j=2}^N c_j(\Delta x_j)$, where $c_j(\Delta x_j) = \Delta x_j^2 + (b_j - b_{j-1} - \Delta x_j)^2$ is a convex function of Δx_j.

The OVD is thus formulated as an integer programming (IP) problem:

$$\begin{aligned} \min\ & \mathcal{E}(\mathbf{x}) = \sum_{j=2}^{N} c_j(x_j - x_{j-1}) = \sum_{j=2}^{N} c_j(\Delta x_j) \\ \text{s.t.}\quad & 0 \le x_j = x_1 + \sum_{k=2}^{j} \Delta x_k \le b_k \qquad \forall j \in [2, N-1] \\ & x_1 = b_1 \qquad x_N = 0 \end{aligned} \tag{3}$$

To use the local searching strategy, we next introduce the concept of L$^\natural$-convexity of an integer function [12,13,14].

Definition 1. *A function f with domain $\mathbb{D}$ is L$^\natural$-convex if and only if $\forall \mathbf{x}, \mathbf{y} \in \mathbb{D}$ and $\alpha \in \mathbb{Z}^+$, $(\mathbf{x} - \alpha\mathbf{1}) \vee \mathbf{y}, \mathbf{x} \wedge (\mathbf{y} + \alpha\mathbf{1}) \in \mathbb{D}$, and*

$$f((\mathbf{x} - \alpha\mathbf{1}) \vee \mathbf{y})) + f(\mathbf{x} \wedge (\mathbf{y} + \alpha\mathbf{1})) \le f(\mathbf{x}) + f(\mathbf{y})$$

where $\vee$ and $\wedge$ stands for component-wise maxima and minima, respectively.

Lemma 1. *In Equation (3), the objective of the OVD problem is L$^\natural$-convex.*

From the definition 1, the L$^\natural$-convexity can be easily verified for Equation (3) due to the convexity of the functions $c_j(\cdot)$.

For an OVD solution $\mathbf{x}^\circ$, define the *upper strip* $\mathbb{S}_{up}(\mathbf{x}^\circ)$ (resp., *lower strip* $\mathbb{S}_{lw}(\mathbf{x}^\circ)$) of $\mathbf{x}^\circ$ with $\mathbb{S}_{up}(\mathbf{x}^\circ) = \{\mathbf{x} \mid \mathbf{x}^\circ \le \mathbf{x} \le \mathbf{x}^\circ + \mathbf{1}, x_1 = b_1, x_N = 0\}$ (resp., $\mathbb{S}_{lw}(\mathbf{x}^\circ) = \{\mathbf{x} \mid \mathbf{x}^\circ \ge \mathbf{x} \ge \mathbf{x}^\circ - \mathbf{1}, x_1 = b_1, x_N = 0\}$). The L$^\natural$-convexity indicates that the local searching strategy works for the OVD problem [13,11]. The key idea is in the following: Starting with an initial solution $\mathbf{x}^\circ$, keep on searching the upper strip and the lower strip of $\mathbf{x}^\circ$ to find a better solution. We call the search on the upper strip or the lower strip as a local search step. Kolmogorov and Shioura proved that after $\mathcal{O}(U)$ local search steps, the algorithm terminates and can find the optimal solution to the OVD problem.

Let us consider the local search step on the upper strip $\mathbb{S}_{up}(\mathbf{x}^\circ)$. Note that each element x_j of $\mathbf{x}$ in $\mathbb{S}_{up}(\mathbf{x}^\circ)$ corresponds to exactly the node $v_j(x_j) \in Col(j)$ in H. Thus, the upper strip $\mathbb{S}_{up}(\mathbf{x}^\circ)$ induces a subgraph $H_{\mathbb{S}}$ of H, whose node set is $\{v_1(b_1)\} \cup \bigcup_{j=2}^{N-1} \{v_j(x_j), v_j(x_j+1)\} \cup \{v_N(0)\}$. Computing an optimal solution $\mathbf{x}'$ in $\mathbb{S}_{up}(\mathbf{x}^\circ)$ is equivalent to find a shortest s-to-t (i.e., $v_1(b_1)$-to-$v_N(0)$) path in the subgraph $H_{\mathbb{S}}$, which obviously can be done in $\mathcal{O}(N)$ time. The local search on the lower strip $\mathbb{S}_{up}(\mathbf{x}^\circ)$ can be done in the same way.

Lemma 2. *Given a non-negative integer vector $\mathbf{b} = (b_1, b_2, \ldots, b_N)$, the OVD problem can be solved in $\mathcal{O}(NU)$ time, where $U = \max_{j=1}^{N} b_j$.*

Note that the weights of the edges between two adjacent columns $Col(j)$ and $Col(j+1)$ in H satisfy the Monge property [1]. Thus, applying the matrix searching technique [1], we can also obtain an $\mathcal{O}(NU)$ time algorithm for solving the OVD problem. Although the OVD algorithm based on the local searching

technique outperforms that based on the straightforward s-to-t shortest path algorithm by a factor of $O(U)$, it is still a pseudo-polynomial time algorithm. We next show how to apply the proximity scaling technique introduced by Hochbaum et al. [7] to achieve a polynomial time OVD algorithm.

Proximity scaling

The proximity scaling technique is used to solve separable convex IP problems with linear constraints whose coefficient matrix has a bounded absolute value for each of its subdeterminants [7]. To solve an integer programming problem (IP), a sequence of scaled versions of the integer programming problem (IP-s) need to be solved. We take the objective function $\mathcal{E}(\mathbf{x}) = \sum_{j=2}^{N} c_j(\Delta x_j)$ in Equation (3) which is separable convex. Since all coefficient of Δx_j in Equation (3) are 1, and the coefficients of Δx_j form a triangular matrix with all 1's, there is no submatrix with absolute value of a determinant larger than 1. Thus, the matrix is totally unimodular with the largest absolute value of the subdeterminants $\Delta_A = 1$.

Lemma 3. *The coefficient matrix of Δx_j in Equation (3) is totally unimodular.*

Thus, we can apply the proximity scaling technique for solving the OVD problem. In each scaling iteration we assume that every Δx_j is a multiple of s, i.e., $\Delta x_j = \lambda_j s$, where λ_j is some non-negative integer for each j. If we obtain an optimal solution $\Delta \mathbf{x}^s$ to this OVD problem with the scaling factor of s (denoted by OVD-s), then there exists an optimal solution $\Delta \mathbf{x}$ to the original OVD problem that is within a distance $(N-2)s$ away from $\Delta \mathbf{x}^s$, that is, $||\Delta \mathbf{x}^s - \Delta \mathbf{x}^1||_\infty \leq (N-2)s\Delta_A = (N-2)s$ [7]. That means in the next iteration, we only need to search the strip $\mathbb{S}_d = \{\Delta \mathbf{x} \mid \Delta \mathbf{x}^s - (N-2)s \leq \Delta \mathbf{x} \leq \Delta \mathbf{x}^s + (N-2)s\}$ for the optimal solution to the original OVD problem. When s reaches 1, we obtain the optimal solution $\Delta \mathbf{x}^*$ to the OVD problem.

To solve the OVD-s problem, note that a solution $\Delta \mathbf{x}$ can be obtained by a linear transformation on the corresponding solution $\mathbf{x}$, that is, $\forall j \in [2, N-1], \Delta x_j = x_j - x_{j-1}$. Hence, we can solve the OVD-s problem using the graph model with the local searching technique to obtain an optimal solution, denoted by $\mathbf{x}^s$; then transform it to the solution $\Delta \mathbf{x}^s$. Note that $||\Delta \mathbf{x}^s - \Delta \mathbf{x}^1||_\infty \leq (N-2)s$, where $\Delta \mathbf{x}^1$ is the vector the linear transformation ($\Delta x_j = x_j - x_{j-1}$) of the optimal solution $\mathbf{x}^1$ to the original OVD problem. With the reverse transformation $x_j = x_1 + \sum_{k=2}^{j} \Delta x_k$, we have $||\mathbf{x}^s - \mathbf{x}^1||_\infty \leq (N-2)^2 s$, where $\mathbf{x}^1$ is an optimal solution to the original OVD problem. This indicates that in our graph model, we need to search the strip $\mathbb{S}_g = \{\mathbf{x} \mid \mathbf{x}^s - (N-2)^2 s \leq \mathbf{x} \leq \mathbf{x}^s + (N-2)^2 s\}$ for each scale s. Recall that in the OVD-s problem, we assume that the solution is a multiple of s. Thus, we resample the strip $\mathbb{S}_g$ with a factor of s, yielding a corresponding subgraph H^s of H. The number of nodes in H^s is $\mathcal{O}(N^3)$, with each of the N columns having $\mathcal{O}(N^2)$ nodes. Applying our local searching strategy, an optimal solution $\mathbf{x}^s$ to the OVD-s problem can be found in $\mathcal{O}(N^3)$ time. For our OVD-s problem, we start with the initial scale $s = \frac{U}{4(N-2)^2}$, where U is the maximum intensity level of $\mathbf{b}$. Then, $||\mathbf{x}^s - \mathbf{x}^1||_\infty \leq \frac{U}{4}$. Hence, the scaling algorithm terminates in $\mathcal{O}(\log U)$ iterations.

Lemma 4. *Given a non-negative integer vector* $\mathbf{b} = (b_1, b_2, \ldots, b_N)$*, the OVD problem can be solved in* $\mathcal{O}(N^3 \log U)$ *time, where* $U = \max_{j=1}^{N} b_j$*.*

Up to this point, the OSF-TV problem can be solved, as follows. The optimal set of $(d-1)$ feathering regions can be obtained using the shortest path model in Section 2.1 to compute a shortest u_1-to-t path in G. The graph G has $(n+1)$ nodes and $O(n\varpi)$ edges. Based on Lemma 4, the weight of each edge in G can be computed in $\mathcal{O}(m\varpi^3 \log U)$ time since IM A has m rows, where U is the maximum intensity level of A. Thus, the optimal set of feathering regions can be computed in $\mathcal{O}(mn\varpi^4 \log U)$ time. Note that each row of the resulting feathering regions can be decomposed by Lemma 4, yielding a split of the IM A with d sub-IMs, whose total sum of the complexity is minimized.

Lemma 5. *Given an instance of the OSF-TV problem, an optimal split can be obtained in* $\mathcal{O}(mn\varpi^4 \log U)$ *time.*

2.3 Accelerating the Computation of the Shortest u_1-to-t Path in G by Monge Property

In this section, we exploit the Monge property [1] of the graph $G = (V, E)$ define in Section 2.1. This property enables us to compute a shortest u_1-to-t path in G by only examining a small portion of the edge set E, yielding an $\mathcal{O}(mn\varpi^2\alpha(\varpi) \log U)$ time algorithm, where $\alpha(\cdot)$ is the inverse Ackermann function.

Lemma 6. *Given four nodes* $u_i, u_{i+1} \in L_k$ *and* $u_j, u_{j+1} \in L_{k+1}$ *in* G *with* $1 < k < d$*,* $w(u_i, u_j) + w(u_{i+1}, u_{j+1}) \leq w(u_i, u_{j+1}) + w(u_{i+1}, u_j)$*.*

Proof. First, if either (u_i, u_{j+1}) or (u_{i+1}, u_j) is not an edge in G, then we can assume that the weight $w(\cdot, \cdot)$ of a non-existent edge is $+\infty$ and the lemma holds. We thus consider the case that both (u_i, u_{j+1}) and $w(u_{i+1}, u_j)$ are edges in G, which also indicates that both (u_i, u_j) and $w(u_{i+1}, u_{j+1})$ are graph edges. Hence, we only need to prove the lemma assuming all (u_i, u_j), (u_{i+1}, u_{j+1}), (u_i, u_{j+1}), and (u_{i+1}, u_j) are edges in G.

Note that the edge cost is defined as the increase of the complexity due to the decomposition $(F_k^{(0)}, F_k^{(1)})$ of the feathering region F_k, that is, $R(F_k^{(0)}) + R(F_k^{(1)}) - R(F_k)$. The part of $R(F_k^{(0)}) + R(F_k^{(1)})$ can be computed by the OVD algorithm in Section 2.2, and the part $R(F_k)$ is invariant with the decomposition. We next examine $R(F_k^{(0)}) + R(F_k^{(1)})$ and $R(F_k)$, separately. Decompose the edge weight $w(u_i, u_j)$ into $w_1(u_i, u_j) - w_2(u_i, u_j)$, where $w_1(u_i, u_j)$ is the total complexity induced by the decomposition of the corresponding feathering region, i.e. $R(A[j \,..\, i + \varpi - 1]^{(0)}) + R(A[j \,..\, i + \varpi - 1]^{(1)})$, and $w_2(u_i, u_j)$ is the original complexity of $A[j \,..\, i + \varpi - 1]$, i.e., $R(A[j \,..\, i + \varpi - 1])$.

Recall that the weight $w_1(u_i, u_j)$ can be computed by applying the OVD algorithm on each extended row of the feathering region $A[j \,..\, i + \varpi - 1]$, separately. Thus, it is sufficient to assume that A consists of only one row.

In the graph model for the OVD problem, the optimal decomposition of a feathering region of A can be represented by a shortest path from $v_{i-1}(b_{i-1})$

to $v_{j+1}(0)$. As illustrated in Fig. 1, assume that the dashed line (the dotted parts stands for omissions in the intermediate columns) is a shortest path from $v_{i-1}(b_{i-1})$ to $v_{j+2}(0)$, thus the weight of this path is $w_1(u_i, u_{j+1})$. We also assume that the solid line is a shortest path from $v_i(b_i)$ to $v_{j+1}(0)$, thus the weight of this path is $w_1(u_{i+1}, u_j)$. Denote the dashed path by $p_b = v_{i-1}(b_{i-1}) \rightarrow v_i(x_i) \rightsquigarrow \cdots \rightsquigarrow v_{j+1}(x_{j+1}) \rightarrow v_{j+2}(0)$, where $x_i \leq b_i$ and $x_{j+1} \geq 0$, and denote the solid path by $p_r = v_i(b_i) \rightsquigarrow \cdots \rightsquigarrow v_{j+1}(0)$. We then distinguish two cases.

1. If $x_i = b_i$ and $x_{j+1} = 0$, then both path-ends of the solid path p_r are on the dashed path p_b. We can simply concatenate $v_{i-1}(b_{i-1}) \rightarrow v_i(x_i)$ and p_r together to form a $v_{i-1}(b_{i-1})$-to-$v_{j+1}(0)$ path, and pick the sub-path of p_b without the first line segment $v_{i-1}(b_{i-1}) \rightarrow v_i(x_i)$ as a $v_i(b_i)$-to-$v_{j+2}(0)$ path. It is clear that the total weight of the two new paths is the same as the total weight of the dashed and the solid paths. Since $w_1(u_i, u_j)$ and $w_1(u_{i+1}, u_{j+1})$ are the weights of the two shortest paths from $v_{i-1}(b_{i-1})$ to $v_{j+1}(0)$ and from $v_i(b_i)$ to $v_{j+2}(0)$, respectively, we have

$$w_1(u_i, u_j) + w_1(u_{i+1}, u_{j+1}) \leq w_1(u_i, u_{j+1}) + w_1(u_{i+1}, u_j)$$

2. Otherwise, there must exists some locations where the dashed path and the solid path across (or overlap) each other, as shown in Fig. 1. In this case, we can introduce two new edges to uncross the dashed and the solid paths, yielding two new paths $v_{i-1}(b_{i-1})$-to-$v_{j+1}(0)$ and $v_i(b_i)$-to-$v_{j+2}(0)$, as shown with the gray thick lines in Fig. 1. More precisely, assume that the crossing (overlapping) happens at columns $Col(l)$ and $Col(l+1)$, the dashed line segment is $v_l(x_l) \rightarrow v_{l+1}(x_{l+1})$ and the solid line segment is $v_l(x'_l) \rightarrow v_{l+1}(x'_{l+1})$, where $x_l \leq x'_l$ and $x_{l+1} \geq x'_{l+1}$. Since the edge weights are convex with respect to Δx

$$\begin{aligned} &w_l(v_l(x_l), v_{l+1}(x'_{l+1})) + w_l(v_l(x'_l), v_{l+1}(x_{l+1})) \\ \leq &w_l(v_l(x_l), v_{l+1}(x_{l+1})) + w_l(v_l(x'_l), v_{l+1}(x'_{l+1})) \end{aligned}$$

   Again, Since $w_1(u_i, u_j)$ and $w_1(u_{i+1}, u_{j+1})$ are the weights of the two shortest paths from $v_{i-1}(b_{i-1})$ to $v_{j+1}(0)$ and from $v_i(b_i)$ to $v_{j+2}(0)$, respectively, thus

$$w_1(u_i, u_j) + w_1(u_{i+1}, u_{j+1}) \leq w_1(u_i, u_{j+1}) + w_1(u_{i+1}, u_j)$$

For the $w_2(u_i, u_j) = R(A[u_i..u_j]) = \sum_{t=1}^{m} \sum_{k=i}^{j+1} (a_{t,k} - a_{t,k-1})^2$, it is not hard to verify that

$$w_2(u_i, u_j) + w_2(u_{i+1}, u_{j+1}) = w_2(u_i, u_{j+1}) + w_2(u_{i+1}, u_j). \tag{4}$$

Therefore, we have

$$w(u_i, u_j) + w(u_{i+1}, u_{j+1}) \leq w(u_i, u_{j+1}) + w(u_{i+1}, u_j) \tag{5}$$

This proves the lemma. □

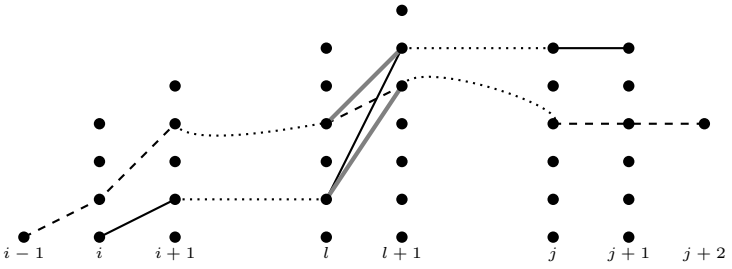


Fig. 1. Illustrating the Monge property. Note that on the same column $Col(j)$, the higher the node, the smaller the corresponding x_j. The bottom node on column $Col(j)$ is $v_j(b_j)$, and the top one is $v_j(0)$.

The Monge property as shown in Lemma 6 can be used to speed up the computation of the shortest u_1-to-t path in G. For each node u_j in the k-th layer L_k, denote by $sw_k(j)$ the shortest path weight of u_1-to-u_j. Clearly, $sw_k(j) = \min\{sw_{k-1}(j') + w(u_{j'}, u_j) | u_{j'} \in L_{k-1}, \varpi - \Delta \leq j - j' \leq \varpi - \delta\}$. Note that $(u_{j'}, u_j)$ is an edge in E if and only if $u_{j'} \in L_{k-1}$, $u_j \in L_k$, and $\varpi - \Delta \leq j - j' \leq \varpi - \delta$. Thus, all the outgoing edges of each node $u_{j'}$ and the incoming edges of each u_j can be represented implicitly. In addition, any edge in G can be accessed in $\mathcal{O}(1)$ time and its weight can be computed in $\mathcal{T} = \mathcal{O}(m\omega^3 \log U) = \mathcal{O}(m\varpi^3 \log U)$ time, where U is the maximum intensity level of A. The Monge property is normally defined on matrices [1]. We consider the matrix M_k that contains the path weight $sw_{k-1}(j') + w(u_{j'}, u_j)$ for every edge $(u_{j'}, u_j)$ between nodes in two consecutive layers L_k and L_{k+1} of G, where $1 < k < d$. Lemma 6 actually indicates that M_k is a staircase matrix with concave Monge property [1,10]. Hence, by using the staircase Monge matrix searching technique [10], it takes $\mathcal{O}(\mathcal{T}\varpi\alpha(\varpi))$ time to compute all shortest paths from u_1 to all nodes in L_k when knowing all $sw_{k-1}(j')$ of L_{k-1}, where $\alpha(\cdot)$ is the inverse Ackermann function. Considering all d layers, a shortest u_1-to-t path can be computed in $\mathcal{O}(dm\varpi^3 \log U\alpha(\varpi)) = \mathcal{O}(mn\varpi^2\alpha(\varpi) \log U)$ time.

Theorem 1. *Given an IM A of size $m \times n$, an integer maximum allowable field width ϖ, and the feathering region width range $0 < \delta < \Delta < \varpi$, the optimal field splitting with minimized total variation (OFS-TV) problem can be solved in $\mathcal{O}(mn\varpi^2\alpha(\varpi) \log U)$ time, where U is the maximum intensity level of A.*

References

1. Aggarwal, A., Klawe, M., Moran, S., Shor, P., Wilber, R.: Geometric applications of a matrix-searching algorithm. Algorithmica 2(1), 195–208 (1987)
2. Bortfeld, T., Craft, D., Halabi, T., Trofimov, A., Monz, M., Küfer, K.: Quantifying the Tradeoff Between Complexity and Conformality. Medical Physics 32, 2085 (2005)

3. Chen, D., Healy, M., Wang, C., Wu, X.: A New Field Splitting Algorithm for Intensity-Modulated Radiation Therapy. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, p. 4. Springer, Heidelberg (2007)
4. Chen, D., Hu, X., Luan, S., Wang, C.: Geometric algorithms for static leaf sequencing problems in radiation therapy. In: Proceedings of the nineteenth annual symposium on Computational geometry, pp. 88–97. ACM, New York (2003)
5. Chen, D., Wang, C.: Field splitting problems in intensity-modulated radiation therapy. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, p. 690. Springer, Heidelberg (2006)
6. Fowler, J., Welsh, J., Howard, S.: Loss of biological effect in prolonged fraction delivery. International journal of radiation oncology, biology, physics 59(1), 242–249 (2004)
7. Hochbaum, D., Shanthikumar, J.: Convex separable optimization is not much harder than linear optimization. Journal of the ACM (JACM) 37(4), 843–862 (1990)
8. Kamath, S., Sahni, S., Li, J., Palta, J., Ranka, S.: A Generalized Field Splitting Algorithm for Optimal IMRT Delivery Efficiency. Medical Physics 32, 1890 (2005)
9. Kamath, S., Sahni, S., Palta, J., Ranka, S.: Algorithms for optimal sequencing of dynamic multileaf collimators. Physics in Medicine and Biology 49(1), 33–54 (2004)
10. Klawe, M., Kleitman, D.: An almost linear time algorithm for generalized matrix searching. SIAM Journal on Discrete Mathematics 3, 81 (1990)
11. Kolmogorov, V., Shioura, A.: New Algorithms for the Dual of the Convex Cost Network Flow Problem with Application to Computer Vision. Mathematical Programming (2007) (Submitted)
12. Murota, K.: Discrete convex analysis. Mathematical Programming 83(1), 313–371 (1998)
13. Murota, K.: Discrete Convex Analysis: Monographs on Discrete Mathematics and Applications, vol. 10. Society for Industrial and Applied Mathematics, Philadelphia (2003)
14. Murota, K.: On steepest descent algorithms for discrete convex functions. SIAM Journal on Optimization 14(3), 699–707 (2004)
15. Oldham, M., Rowbottom, C.: Radiosurgery with a motorized micro-multileaf-collimator: initial experiences in planning and commissioning. In: Proceedings of the 43rd Annual Meeting of the American Association of Physicists in Medicine, AAPM (2001)
16. Süss, P., Küfer, K.: Smooth intensity maps and the Bortfeld-Boyer sequencer. Berichte des Fraunhofer-Instituts für Techno- und Wirtschaftsmathematik Report 109 (2007)
17. Webb, S.: Intensity-modulated radiation therapy. Inst. of Physics Pub. Inc. (2001)
18. Wu, Q., Arnfield, M., Tong, S., Wu, Y., Mohan, R.: Dynamic splitting of large intensity-modulated fields. Physics in Medicine and Biology 45, 1731–1740 (2000)
19. Wu, X.: Efficient algorithms for intensity map splitting problems in radiation therapy. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, p. 504. Springer, Heidelberg (2005)
20. Wu, X., Dou, X., Bayouth, J., Buatti, J.: New algorithm for field splitting in radiation therapy. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 692–703. Springer, Heidelberg (2007)

Approximation Schemes for Scheduling with Availability Constraints*

Bin Fu[1], Yumei Huo[2], and Hairong Zhao[3]

[1] Department of Computer Science, University of Texas–Pan American, Edinburg, TX 78539, USA
binfu@cs.panam.edu
[2] Department of Computer Science, College of Staten Island, CUNY, Staten Island, New York 10314, USA
huo@mail.csi.cuny.edu
[3] Department of Mathematics, Computer Science & Statistics, Purdue University Calumet, Hammond, IN 46323, USA
hairong@calumet.purdue.edu

Abstract. We investigate the problems of scheduling n weighted jobs to m identical machines with availability constraints. We consider two different models of availability constraints: the preventive model where the unavailability is due to preventive machine maintenance, and the fixed job model where the unavailability is due to a priori assignment of some of the n jobs to certain machines at certain times. Both models have applications such as turnaround scheduling or overlay computing. In both models, the objective is to minimize the total weighted completion time. We assume that m is a constant, and the jobs are non-resumable. For the preventive model, it has been shown that there is no approximation algorithm if all machines have unavailable intervals even when $w_i = p_i$ for all jobs. In this paper, we assume there is one machine permanently available and the processing time of each job is equal to its weight for all jobs. We develop the first PTAS when there are constant number of unavailable intervals. One main feature of our algorithm is that the classification of large and small jobs is with respect to each individual interval, thus not fixed. This classification allows us (1) to enumerate the assignments of large jobs efficiently; (2) and to move small jobs around without increasing the objective value too much, and thus derive our PTAS. Then we show that there is no FPTAS in this case unless $P = NP$.

For fixed job model, we first show that if job weights are arbitrary then there is no constant approximation for a single machine with 2 fixed jobs or for two machines with one fixed job on each machine, unless $P = NP$. As the preventive model, we assume that the weight of a job is the same as its processing time for all jobs. We show that the PTAS for the preventive model can be extended to solve this problem when the number of fixed jobs and the number of machines are both constants.

* This research is supported by NSF Career Award 0845376.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 77–88, 2010.

1 Introduction

In this paper we study the problems of scheduling n weighted jobs to m identical machines with the objective of minimizing total weighted completion time subject to availability constraints. We consider two different models of availability constraints: the preventive model where the unavailability is due to preventive machine maintenance, and the fixed job model where the unavailability is due to a priori assignment of some of the n jobs to certain machines at certain times. Both models have applications such as turnaround scheduling or overlay computing ([4]). In both models, we assume that m is a constant, and the jobs are non-resumable, i.e., if interrupted by an unavailable interval, a job has to be restarted after the machine becomes available.

Let us first introduce some notations. For convenience, we use $1, 2, \cdots, n$ to denote the jobs. Each job i has a processing time p_i and a weight w_i. Given a schedule S of the jobs, the completion time of job i in S is denoted by $C_i(S)$. If S is clear from the context, we will use C_i for short. The total weighted completion time is denoted by $F_w(S) = \sum w_i C_i(S)$. The goal is to schedule the set of jobs on one or more parallel machines so as to minimize $F_w(S)$. For single machine with k unavailable intervals due to preventive maintenance, our problem is denoted by $1, h_k \mid nr-a \mid \sum w_i C_i$. For parallel machine environment, let $M = \{M_0, M_1, M_2, \cdots, M_m\}$ be a set of $m+1$ parallel machines, where machine M_0 is always available and machines $M_1, M_2, \cdots, M_m$ have $k_1, k_2, \cdots, k_m$ unavailable intervals, respectively. Then our problem is denoted by $P_{1,m}, h_{k_1}, h_{k_2}, \cdots, h_{k_m} \mid nr-a \mid \sum w_i C_i$. If the unavailable intervals are due to fixed jobs, the problems are denoted as $1, h_c \mid nr-a,$ fixed $\mid \sum w_i C_i$ for single machine with c fixed jobs and $P_{0,m}, h_{k_1}, h_{k_2}, \cdots, h_{k_m} \mid nr-a,$ fixed $\mid \sum w_i C_i$ for m machines with $k_1, k_2, \cdots, k_m$ fixed jobs respectively.

Literature Review. The preventive model has been studied a lot in the literature, see for example [2], [10], [11], [12], [15]. We review some of the related results here. For more information, please refer to the surveys by Saidy et. al ([18]), Schmidt([20]), Lee ([14]) and the references therein. When the machines are always available, the single machine scheduling problem, denoted as $1 \mid\mid w_i C_i$, can be solved optimally by the WSPT (Weighted Shortest Processing time) algorithm which schedules the jobs in the nondecreasing order of p_i/w_i ([17]). The problem becomes NP-hard when there are multiple machines. The problem $1, h_1 \mid nr-a \mid \sum w_i C_i$ is studied in [1] and [13] and is shown to be NP-hard in the ordinary sense. Kellerer and Strusevish ([9]) proposed a 4-approximation for $1, h_1 \mid nr-a \mid \sum w_i C_i$ and an FPTAS. Kacem and Mahjoub ([7]) and Fu et al. [6] subsequently developed FPTASs to improve the time complexity. For the multiple machine environment, Kaspi and Montreuil ([8]) and Liman ([16]) studied the case where the jobs are unweighted, and each machines only has a single unavailable interval starting at time 0. If the jobs are weighted, Lee ([12]) provided dynamic programming for $P_{1,1}, h_1 \mid nr-a \mid \sum w_i C_i$. Fu et al. [6] showed that there is no polynomial time algorithm that approximates the optimal solution to $P_{0,2}, h_1, h_1 \mid nr-a, w_i = p_i \mid \sum w_i c_i$ within an exponential

factor and they developed an FPTAS when there is only one unavailable interval among all machines and jobs have arbitrary weight.

Scheduling with fixed job model was studied by Scharbrodt, Steger and Weisser in [19]. They studied the makespan minimization problem and presented a PTAS when m is a constant. Approximation algorithm was also developed in [5]. Scharbrodt et. al also proved that when m is arbitrary, there is no approximation algorithm with ratio $3/2 - \epsilon$ unless $P = NP$. In 2009, Diedrich and Jansen ([4]) complemented this negative result by giving a $3/2 + \epsilon$ approximation algorithm. So far no result is known about the total weighted completion time in this model.

New Contributions. For the preventive model, we derive a PTAS for the problem $P_{1,m},\ h_{k_1},\ h_{k_2},\ \cdots,\ h_{k_m} \mid nr - a, w_i = p_i \mid \sum w_i C_i$ and show that no FPTAS exists for this problem. For the fixed job model, we first show that with arbitrary job weights, no constant approximation exists for $1, h_2 \mid \text{ fixed } \mid \sum w_i C_i$ or $P_{0,2}, h_1, h_1 \mid \text{ fixed } \mid \sum w_i C_i$. We then extend our PTAS for preventive model to solve the problem $P_{0,m}, h_{k_1}, h_{k_2}, \cdots, h_{k_m} \mid nr - a, \text{ fixed }, w_i = p_i \mid \sum w_i C_i$.

It is tempting to compare the complexity between preventive model and fixed job model. It has been shown that there is no PTAS for $P_{0,m}, h_{k_1}, h_{k_2}, \cdots, h_{k_m} \mid nr - a, w_i = p_i \mid \sum w_i C_i$ in the preventive model. This is the reason that we require at least one machine is permanently available; on the other hand, for fixed job model, assuming $w_i = p_i$ for all jobs, the PTAS works even when all machines have unavailable intervals. In this sense, the problems in the fixed job model are somehow easier than the problems in the preventive model. However, the result of no constant approximation mentioned above shows that for the general case of arbitrary job weight, the problems in fixed job model are as difficult as the problems in the preventive model.

One technical contribution of this paper lies in the PTAS design, where the jobs are classified as large jobs and small jobs with respect to each individual interval. That is, different intervals may define different sets of large jobs and small jobs. This method allows us (1) to enumerate the assignment of large jobs efficiently; (2) to move the small jobs around without increasing the objective value too much, and thus derive our PTAS. This may give some insights for other related problems or performance criteria.

2 Preventive Model

In this section, we study the case that the machine unavailability is due to preventive maintenance. We first describe our main results, a PTAS for $P_{1,m}, h_{k_1}, h_{k_2}, \cdots, h_{k_m} \mid nr - a, w_i = p_i \mid \sum w_i C_i$. Then we show this problem doesnot admit FPTAS.

For each machine, its availability can be described as a sequence of alternating available intervals and unavailable intervals, all the intervals are bounded except the last which may be available or unavailable. Let c_1 be the total number of the *bounded available intervals* on machines $M_1,\ M_2,\ \cdots,\ M_m$. We use

$I_1, \cdots, I_{c_1}$ to denote these intervals and their lengths and let s_i be the starting time of interval I_i. It is easy to see that c_1 is bounded by the total number of unavailable intervals $\sum h_{k_i}$. Suppose machine M_0, M_1, $\cdots$ M_{c_2}, $c_2 \leq m$, each has an unbounded available interval, denoted by $I_0^\infty, I_1^\infty, \cdots, I_{c_2}^\infty$, respectively. Let s_i^∞ be the starting time of interval I_i^∞.

Let S be any feasible schedule of the job set $J = \{1, 2, \cdots, n\}$, where $p_i = w_i$ for all i. Schedule S assigns each job to an available interval. To minimize the total weighted completion time, it is sufficient to assume that S does not contain any idle time between the jobs in each interval. Furthermore, the jobs in each available interval can be scheduled in an arbitrary order.

For any set of jobs J, let $P(J)$ be the total processing time of jobs in J; i.e $P(J) = \sum_{i \in J} p_i$. Let $Q(J)$ be the minimum total weighted completion time of jobs in J on a single machine if jobs were continuously scheduled from time 0. Since we assume $p_j = w_j$ for all j, then the order of the jobs in the schedule does not matter, and we always have

$$Q(J) = \sum_{i \in J} p_i^2 + \sum_{i,j \in J}^{i \neq j} p_i p_j \geq \frac{1}{2} \left(\sum_{i \in J} p_i \right)^2 = \frac{1}{2} P(J)^2 \tag{1}$$

Assuming the jobs in the same interval are always scheduled in decreasing order of their length, in this way, a schedule is uniquely determined by the assignment of all the jobs to intervals. In this paper, we will use the *assignment and the schedule interchangeably* unless we explicitly specify. We will use a tuple to represent the job assignment in a schedule (maybe a partial schedule): $(X_1, X_2, \cdots, X_{c_1}, Y_0, Y_1, \cdots, Y_{c_2})$, where $X_i (1 \leq i \leq c_1)$ contains jobs assigned to *bounded available interval* I_i, and $Y_i (0 \leq i \leq c_2)$ contains jobs allocated to *unbounded available interval* I_i^∞. A feasible schedule or assignment is one such that for all available intervals, the total length of the jobs assigned to it is less than the length of the interval. Given an assignment/schedule $(X_1, X_2, \cdots, X_{c_1}, Y_0, Y_1, \cdots, Y_{c_2})$, it is easy to verify that the total weighted completion time of the schedule is:

$$F_w(S) = \left(\sum_{i=1}^{c_1} (Q(X_i) + s_i P(X_i)) \right) + \left(\sum_{i=0}^{c_2} (Q(Y_i) + s_i^\infty P(Y_i)) \right). \tag{2}$$

Treating a schedule as an assignment of the jobs gives us a new perspective of the schedules. To get a schedule of a set of jobs J, we just need to find an assignment of the jobs to the available intervals in J. Let $J = J_1 \cup J_2$. If we have an assignment of jobs in J_1, an assignment of jobs in J_2, then we can simply combine them to get an assignment of all jobs in J. As we will see later, this is one of the main ideas of our algorithm. We will also use the following facts in our analysis later.

$$P(J_1 \cup J_2) = P(J_1) + P(J_2) \tag{3}$$

$$Q(J_1 \cup J_2) = Q(J_1) + Q(J_2) + P(J_1) \cdot P(J_2) \tag{4}$$

2.1 Polynomial Time Approximation Scheme

Given an instance of $P_{1,m}, h_{k_1}, h_{k_2}, \cdots, h_{k_m} \mid nr - a, w_i = p_i \mid \sum w_i C_i$ and a constant $0 < \epsilon < 1$, our goal is to design an efficient algorithm that finds an assignment of the jobs to machines so that the total weighted completion time is at most $(1 + \epsilon)$ times the optimal. Our algorithm consists of four phases.

1. Phase I: Assign some of the large jobs to the bounded available intervals;
2. Phase II: Assign the remaining jobs including the small jobs and the remaining large jobs to all the available intervals;
3. Phase III: Reassign some of the jobs to make the schedule feasible;
4. Phase IV: Search for the best schedule: both Phase I and II have many alternatives, thus result in many candidate schedules, find the one with minimum weighted completion time.

As one can see, the main part of the algorithm lies in the first three phases where we try to assign and reassign jobs. In Phase IV, we simply search for the best solution. In the following, we describe each of the first three phases in detail, and show how each step can be implemented efficiently.

Phase I: Assign Large Jobs to Bounded Available Intervals. First let us define *large job*. Given a constant parameter $0 < \delta < 1$ which depends on ϵ, for each bounded available interval I_i $(1 \leq i \leq c_1)$, we say that a job is a *large job with respect to* I_i if its processing time is greater than or equal to $\delta \cdot I_i$; otherwise, it is a *small job with respect to* I_i. Note that a job may be large with respect to one interval while being small with respect to another.

To assign the large jobs into the bounded available intervals, we use brute force. Specifically, a job can be assigned to I_i only if it is a large job with respect to I_i. We enumerate all the possible assignment of large jobs to bounded intervals such that the total length of jobs assigned to the interval I_i is at most I_i for each $1 \leq i \leq c_1$. The following lemma gives a bound on the number of possible assignments of the large jobs, see the Appendix for proof.

Lemma 1. *There are at most* $O(n^{\frac{c_1}{\delta}})$ *possible assignments of the large jobs, where* $0 < \delta < 1$ *is a constant depending on* ϵ.

Proof. For each bounded available interval I_i, the number of large jobs that can be assigned to I_i is at most $I_i/(\delta I_i) = 1/\delta$. So there are at most $O(n^{\frac{1}{\delta}})$ possible ways to assign large jobs to interval I_i. Since there are c_1 bounded available intervals, there are at most $O(n^{\frac{c_1}{\delta}})$ possible assignments of the large jobs to bounded available intervals in total.

Phase II: Assign Remaining Jobs. Once the large jobs in each interval have been fixed, we use divide and conquer to assign the remaining jobs to all the available intervals which includes c_1 bounded intervals and c_2 unbounded intervals. For this phase, we allow infeasible assignments, that is, the total length

of jobs assigned to the interval is more than the length of the interval. However, we only consider those assignments such that the total length of the large and small jobs assigned to the interval is at most $(1+\delta)$ times the length of the interval, where δ is defined as in Phase I. We say these intervals are *valid* (may be infeasible) with respect to δ. An assignment is valid (feasible) if and only if all its intervals are valid (feasible). For each fixed large job assignment $(u_1^l, \cdots, u_{c_1}^l, \emptyset, \emptyset, \cdots, \emptyset)$, Phase II, as described below, returns a list of assignments of the remaining jobs. Furthermore, if $(u_1^s, \cdots, u_{c_1}^s, v_1, v_2, \cdots, v_{c_2})$ is an assignment in the returned list, then $(u_1, \cdots, u_{c_1}, v_1, v_2, \cdots, v_{c_2})$, $u_j = u_j^l \cup u_j^s$, $1 \le j \le c_1$, is a *valid* assignment of all the jobs.

1. If there is only a single remaining job J_k, it can be assigned to any of the unbounded available intervals, or to those bounded intervals I_i such that J_k is small with respect to I_i, and I_i remains a valid interval with respect to δ, return all these possible assignments of J_k.
2. Otherwise
   (a) divide the remaining jobs into two equal sets J_1 and J_2
   (b) recursively assign the jobs in J_1 and J_2, let $List_1$ and $List_2$ be the returned lists of assignments for J_1 and J_2, respectively.
   (c) for each assignment in $List_1$ and each assignment from $List_2$, combine them to get an assignment of the jobs in $J_1 \cup J_2$.
   (d) filter the assignments:
   Let $f = (1 + \frac{\delta}{4 \log n})$. We say two assignments $(u_1, \cdots, u_{c_1+c_2+1})$ and $(v_1, \cdots, v_{c_1+c_2+1})$ are in the same region with respect to δ if for any $1 \le j \le c_1 + c_2 + 1$, there exist two integers k_1, k_2, such that $f^{k_1-1} \le P(u_j), P(v_j) < f^{k_1}$, and $f^{k_2-1} \le Q(u_j), Q(v_j) < f^{k_2}$.
   To filter the assignments, we keep from each region only one assignment $(u_1^s, \cdots, u_{c_1}^s, v_1, v_2, \cdots, v_{c_2})$ as the representative such that $(u_1^l \cup u_1^s, \cdots, u_{c_1}^l \cup u_{c_1}^s, v_1, v_2, \cdots, v_{c_2})$ is a *valid* assignment. Finally return these representative assignments.

Now we analyze Phase II. We have the following lemma about the relationship between any feasible schedule and the list of assignments returned by the algorithm.

Lemma 2. *Let J be a set of n jobs. For any feasible schedule $S = (u_1, \cdots, u_{c_1}, v_0, v_1, \cdots, v_{c_2})$ of the jobs in J, after Phase II, there exists one valid assignment $(u'_1, \cdots, u'_{c_1}, v'_0, v'_1, \cdots, v'_{c_2})$ of all jobs such that the following properties hold:*

(1) the set of large jobs in u'_j is the same as the set of large jobs in u_j, denoted by u_j^l, for $j = 1, \cdots, c_1$.
(2) $Q(u'^s_j) \le f^{2\log n} \cdot Q(u_j^s)$, $P(u'^s_j) \le f^{\log n} \cdot P(u_j^s)$ for $j = 1, \cdots, c_1$, where u'^s_j and u_j^s are the small jobs in u'_j and u_j respectively.
(3) $Q(v'_j) \le f^{2\log n} \cdot Q(v_j)$, $P(v'_j) \le f^{\log n} \cdot P(v_j)$ for $j = 0, 1, \cdots, c_2$.

Proof. Since we enumerate all possible assignments of the large jobs, after Phase I, we must have one assignment $(u_1^l, \cdots, u_{c_1}^l, \emptyset, \emptyset, \cdots, \emptyset)$. We will consider only

the assignments of the remaining jobs that are based on this assignment. Therefore, property (1) is always true.

Suppose there are n' remaining jobs. When $n' = 0$ or $n' = 1$, the lemma is trivially true. Assuming that the claim is true for $k < n'$, we now verify the hypothesis for n'. In Phase II, the remaining jobs are divided into two sets, J_1 and J_2, which are then recursively assigned. Let $(u_1^1, \cdots, u_{c_1}^1, v_0^1, \cdots, v_{c_2}^1)$ be the assignment of the jobs in J_1 by schedule S. Let $(u_1^2, \cdots, u_{c_1}^2, v_0^2, \cdots, v_{c_2}^2)$ be the assignment of the jobs in J_2 by schedule S. Then $u_i^1 \cup u_i^2 = u_i^s$ and $v_i^1 \cup v_i^2 = v_i$. By the inductive hypothesis, there is one assignment of the jobs in J_1, $(u_1'^1, \cdots, u_{c_1}'^1, v_0'^1, \cdots, v_{c_2}'^1)$, where $u_j'^1$ contains only small jobs with respect to I_j, and has the following two properties.

$$Q(u_j'^1) \le f^{2\log\frac{n'}{2}} \cdot Q(u_j^1),\ P(u_j'^1) \le f^{\log\frac{n'}{2}} \cdot P(u_j^1) \text{ for } j = 1, \cdots, c_1.$$
$$Q(v_j'^1) \le f^{2\log\frac{n'}{2}} \cdot Q(v_j^1),\ P(v_j'^1) \le f^{\log\frac{n'}{2}} \cdot P(v_j^1) \text{ for } j = 0, 1, \cdots, c_2.$$

Similarly, by the inductive hypothesis, there is another assignment of the jobs in J_2, $(u_1'^2, \cdots, u_{c_1}'^2, v_0'^2, \cdots, v_{c_2}'^2)$, where $u_j'^2$ contains only small jobs with respect to I_j, and the following two conditions hold.

$$Q(u_j'^2) \le f^{2\log\frac{n'}{2}} \cdot Q(u_j^2),\ P(u_j'^2) \le f^{\log\frac{n'}{2}} \cdot P(u_j^2) \text{ for } j = 1, \cdots, c_1.$$
$$Q(v_j'^2) \le f^{2\log\frac{n'}{2}} \cdot Q(v_j^2),\ P(v_j'^2) \le f^{\log\frac{n'}{2}} \cdot P(v_j^2) \text{ for } j = 0, 1, \cdots, c_2.$$

Combining the above assignments of jobs in J_1 and J_2, we get an assignment of jobs in $J_1 \cup J_2$, $(\widehat{u'}^s_1, \cdots, \widehat{u'}^s_{c_1}, \widehat{v'}_0, \cdots, \widehat{v'}_{c_2})$, where $\widehat{u'}^s_i = u_i'^1 \cup u_i'^2$, and $\widehat{v'}_i = v_i'^1 \cup v_i'^2$. Then by equations (3) and (4), we have

$$P(\widehat{u'}^s_j) = P(u_j'^1) + P(u_j'^2) \le f^{\log\frac{n'}{2}} P(u_j^1) + f^{\log\frac{n'}{2}} P(u_j^2) = f^{\log\frac{n'}{2}} P(u_j^s)$$

$$\begin{aligned} Q(\widehat{u'}^s_j) &= Q(u_j'^1) + Q(u_j'^2) + P(u_j'^1) \cdot P(u_j'^2) \\ &\le f^{2\log\frac{n'}{2}} \cdot Q(u_j^1) + f^{2\log\frac{n'}{2}} \cdot Q(u_j^2) + f^{\log\frac{n'}{2}} \cdot P(u_j^1) \cdot f^{\log\frac{n'}{2}} \cdot P(u_j^2) \\ &\le f^{2\log\frac{n'}{2}} (Q(u_j^1) + Q(u_j^2) + P(u_j^1) \cdot P(u_j^2)) \\ &= f^{2\log\frac{n'}{2}} Q(u_j^s) \end{aligned}$$

In the same way, we can show that the following properties hold for v_j and $\widehat{v'}_j$:

$$P(\widehat{v'}_j) \le f^{\log\frac{n'}{2}} P(v_j) \quad \text{and} \quad Q(\widehat{v'}_j) \le f^{2\log\frac{n'}{2}} Q(v_j)\ .$$

Next, we show this assignment together with the assignment of the large jobs form a valid assignment of all jobs with respect to δ. That is, we show for every $1 \le i \le c_1$, $P(\widehat{u'}^s_i \cup u_i^l) \le (1+\delta) I_i$. First, for given constant $0 < \delta < 1$, and $f = 1 + \frac{\delta}{4\log n}$, we have

$$f^{2\log n'} \le f^{2\log n} = \left(1 + \frac{\delta}{4\log n}\right)^{2\log n} < (1+\delta)\ .$$

Thus $P(\widehat{u'}_i^s \cup u_i^l) = P(\widehat{u'}_i) + P(u_i^l) \le f^{\log \frac{n'}{2}} P(u_i^s) + P(u_i^l) \le (1+\delta)P(u_i^s) + P(u_i^l) = (1+\delta)(P(u_i^s) + P(u_i^l)) = (1+\delta)P(u_i) \le (1+\delta)I_i$.

On the other hand, if $(\widehat{u'}_1^s, \cdots, \widehat{u'}_{c_1}^s, \widehat{v'}_0, \cdots, \widehat{v'}_{c_2})$ is not returned by the filter step, then another assignment in the same region, $(u'^s_1, \cdots, u'^s_{c_1}, v'_0, v'_1, \cdots, v'_{c_2})$, will be returned. Since they are in the same region, we have

$$P(u'^s_j) \le f \cdot P(\widehat{u'}_j^s) \le f \cdot f^{\log \frac{n'}{2}} P(u_j^s) = f^{\log n'} P(u_j^s) \le f^{\log n} P(u_j^s)$$

$$Q(u'^s_j) \le f \cdot Q(\widehat{u'}_j^s) \le f \cdot f^{2\log \frac{n'}{2}} Q(u_j^s) = f^{2\log n'} Q(u_j^s) \le f^{2\log n} Q(u_j^s)$$

$$P(v'_j) \le f P(\widehat{v'}_j) \le f^{\log n} P(v_j) \quad \text{and} \quad Q(v'_j) \le f Q(\widehat{v'}_j) \le f^{2\log n} Q(v_j) \ .$$

In the same way we can show that $(u'^s_1, \cdots, u'^s_{c_1}, v'_0, v'_1, \cdots, v'_{c_2})$ together with $(u_1^l, \cdots, u_{c_1}^l, \emptyset, \emptyset, \cdots, \emptyset)$ form a valid assignment of all jobs with respect to δ.

The following lemma shows the time complexity of Phase II, the proof is given in the Appendix.

Lemma 3. *For a fixed large job assignment, the running time of Phase II is* $O(n(c_1+c_2)(\log_f P)^{2(c_1+c_2+1)})$ *where* $P = \sum p_i$.

Proof. Let J' be the set of the remaining jobs. Then the total number of regions of the assignments is at most $(\log_f Q(J'))^{c_1+c_2+1}$. Since $Q(J') \le \frac{1}{2}P(J')^2 \le \frac{1}{2}P(J)^2$, the number of regions is at most $O(\log_f P(J))^{c_1+c_2+1}$.

Let $T(n')$ be the running time of Phase II for n' remaining jobs. We can use (c_1+c_2+1) linked lists to represent each assignment. When $n'=1$, we return at most (c_1+c_2+1) assignments, $T(1) = O((c_1+c_2)^2)$. Otherwise, we have two recursive calls, each taking time $T(n'/2)$, and returning a list of at most $\log_f P(J')^{(c_1+c_2+1)}$ assignments. To combine one assignment from $List_1$ and one from $List_2$, we just need to merge (c_1+c_2+1) pairs of linked lists which can be done $O(c_1+c_2)$ in total. We have $T(n') = 2T(n'/2) + O((c_1+c_2)(\log_f P(J'))^{2(c_1+c_2+1)})$. One can easily show that the total computation time is $O(n'(c_1+c_2+1)(\log_f P(J')^{2(c_1+c_2+1)}) = O(n(c_1+c_2)(\log_f P)^{2(c_1+c_2+1)})$.

Phase III: Reassign Small Jobs to Make Schedules Feasible. Let $(u_1, u_2, \cdots, u_{c_1}, v_0, v_1, \cdots, v_{c_2})$ be a valid assignment of all jobs after Phase II. As we mentioned before, this valid assignment may not be a feasible schedule because some jobs have to be scheduled at time when the machine is unavailable. So we want to reallocate some of the jobs, in particular, move some jobs from the bounded available intervals to the unbounded available interval on the first machine. In Phase III as described below, we process each valid assignment obtained after Phase II as below, and finally get a feasible assignment.

Phase III: Reassign Small Jobs

(1) Let $(u_1, u_2, \cdots, u_{c_1}, v_0, v_1, \cdots, v_{c_2})$ be a valid but not feasible assignment of all jobs.

(2) For each valid but not feasible interval I_j, i.e, $I_j < P(u_j) < (1+\delta)I_j$, repeat if a job in u_j is scheduled at or after the time $P(v_0)$, reassign the job to machine M_0, update v_0 and u_j.
(3) For each valid but infeasible interval I_j after the above step, reassign some small jobs with total length at most $2\delta \cdot I_j$ to machine M_0, so it becomes feasible

Lemma 4. *Let $S = (u_1, u_2, \cdots, u_{c_1}, v_0, v_1, \cdots, v_{c_2})$ be a valid but infeasible assignment of all jobs after Phase II, and let $S' = (u'_1, u'_2, \cdots, u'_{c_1}, v'_0, v_1, \cdots, v_{c_2})$ be the feasible assignment after Phase III. Then we must have $F_w(S') \leq \frac{8\delta c_1^2}{(1-\delta)^2} F_w(S)$.*

Proof. Let $(\hat{u_1}, \hat{u_2}, \cdots, \hat{u_{c_1}}, \hat{v_0}, v_1, \cdots, v_{c_2})$ be the assignment after step (2), and $\hat{S}$ be the corresponding schedule. First note that if a job is reassigned in step (2), its completion time will not be increased, thus we have $F_w(\hat{S}) \leq F_w(S)$. Let I_j be an infeasible interval (s_j, t_j) after step (2). We assume all jobs are scheduled in decreasing order. Then the last job must be a small job, whose length is at most δI_i. Furthermore, it starts before $P(\hat{v_0})$, finishes after t_j but before $t_j + \delta I_j$. So we must have $I_j < t_j < P(\hat{v_0}) + \delta I_j$, i.e. if I_j is infeasible after step (2), then $I_j \leq \frac{P(\hat{v_0})}{1-\delta}$.

In step (3), some small jobs are reassigned. The order of these reassigned jobs does not matter, but for ease of analysis, we assume these jobs are all inserted from time 0, first the jobs from $\hat{u_0}$, then the jobs from $\hat{u_1}$, and so on.

Next, we analyze the increase of the total weighted completion time. We first analyze the jobs on machine M_0 in $\hat{S}$. Let $I_{\max}$ be the largest infeasible interval. Since the total length of the inserted jobs is at most $(2\delta I_1 + 2\delta I_2 + \ldots 2\delta I_{c_1}) \leq 2\delta c_1 I_{\max}$, for each job in $\hat{v_0}$, its completion time is increased by at most $2\delta c_1 I_{\max}$. Using the fact $I_{\max} \leq \frac{P(\hat{v_0})}{1-\delta}$ and Equation (1), the total weighted completion time of all jobs from $\hat{v_0}$ is increased by at most

$$\sum_{j \in \hat{v_0}} w_j \cdot (2\delta c_1 I_{\max}) = P(\hat{v_0})(2\delta c_1 I_{\max}) \leq \frac{2\delta c_1 P(\hat{v_0})^2}{1-\delta} \leq \frac{2\delta c_1 \cdot 2Q(\hat{v_0})}{1-\delta} = \frac{4\delta c_1 Q(\hat{v_0})}{1-\delta}.$$

For the reassigned small jobs from $\hat{u_1}$, the total weighted completion time is not increased since the completion time of each job is not increased. For the jobs from $\hat{u_2}$, they are preceded by those reassigned small jobs from $\hat{u_1}$, the completion time of each job is increased by at most $2\delta I_1$, thus the total increase of the total weighted completion time is at most $2\delta I_1 \cdot 2\delta I_2 \leq 4\delta^2 I_{\max}^2$. Similarly, for the jobs from $\hat{u_k}$, the total increase of the total weighted completion time is at most $4\delta^2(k-1)I_{\max}^2$. Summing this up for all intervals, the total increase is at most $\sum_{k=1}^{c_1} 4\delta^2(k-1)I_{\max}^2 \leq 2\delta^2 c_1^2 I_{\max}^2 \leq \frac{2\delta^2 c_1^2 P(\hat{v_0})^2}{(1-\delta)^2} \leq \frac{4\delta^2 c_1^2 Q(\hat{v_0})}{(1-\delta)^2}$.

In summary, the total increase from step (3) is at most

$$\frac{4\delta c_1 \cdot Q(\hat{v_0})}{1-\delta} + \frac{4\delta^2 c_1^2 Q(\hat{v_0})}{(1-\delta)^2} \leq \frac{8\delta c_1^2 Q(\hat{v_0})}{(1-\delta)^2} \leq \frac{8\delta c_1^2}{(1-\delta)^2} F_w(\hat{S}) \leq \frac{8\delta c_1^2}{(1-\delta)^2} F_w(S) \ .$$

Lemma 2 and Lemma 4 ensure us the assignment returned by Phase IV is feasible and has a cost close to that of the optimal schedule, see below.

Lemma 5. *For any instance of* $P_{1,m}, h_{k_1}, h_{k_2}, \cdots, h_{k_m} \mid nr - a, w_i = p_i \mid \sum w_i C_i$, *and any constant* ϵ, $0 < \epsilon < 1$, *let* S^* *be the optimal schedule. Let* δ *be a constant such that* $\frac{(16c_1^2+1)\delta}{(1-\delta)^2} \leq \epsilon$ *and* S *be the schedule returned after Phase IV. Then,* $F_w(S) \leq (1+\epsilon)F_w(S^*)$.

Proof. Let $S^* = (u_1^*, \cdots, u_{c_1}^*, v_0^*, v_1^*, \cdots, v_{c_2}^*)$ denote the optimal schedule. By Lemma 2, there exists a valid assignment $S' = (u_1', \cdots, u_{c_1}', v_0', v_1', \cdots, v_{c_2}')$ after Phase II such that the three properties in Lemma 2 hold. By Lemma 4, after Phase III, we obtain a new feasible assignment S'' such that $F_w(S'') \leq (1 + \frac{8{c_1}^2\delta}{(1-\delta)^2})F_w(S')$. Finally in Phase IV, we get a feasible schedule S such that $F_w(S) \leq F_w(S'') \leq (1 + \frac{8{c_1}^2\delta}{(1-\delta)^2})F_w(S')$. By Equation (2),

$$F_w(S') = \left(\sum_{i=1}^{c_1}(Q(u_i') + s_i P(u_i'))\right) + \left(\sum_{i=0}^{c_2}(Q(v_i') + s_i^{\infty} P(v_i'))\right).$$

By properties of Lemma 2, we have $Q(u_j') \leq f^{2\log n} \cdot Q(u_j^*)$ and $P(u_j') \leq f^{\log n} \cdot P(u_j^*)$ for $j = 1, \cdots, c_1$, and $Q(v_j') \leq f^{2\log n} \cdot Q(v_j^*)$, $P(v_j') \leq f^{\log n} \cdot P(v_j^*)$ for $j = 0, 1, \cdots, c_2$. Therefore,

$$\begin{aligned}
F_w(S) &\leq \left(1 + \tfrac{8{c_1}^2\delta}{(1-\delta)^2}\right)\Big(\sum_{i=1}^{c_1}\left(f^{2\log n} \cdot Q(u_i^*) + s_i f^{\log n} \cdot P(u_i^*)\right) \\
&\quad + \sum_{i=0}^{c_2}\left(f^{2\log n} \cdot Q(v_i^*) + s_i^{\infty} f^{\log n} \cdot P(v_i^*)\right)\Big) \\
&= \left(1 + \tfrac{8{c_1}^2\delta}{(1-\delta)^2}\right) f^{2\log n}\left(\sum_{i=1}^{c_1}(Q(u_i^*) + s_i P(u_i^*)) + \sum_{i=0}^{m}(Q(v_i^*) + s_i^{\infty} P(v_i^*))\right) \\
&= \left(1 + \tfrac{8{c_1}^2\delta}{(1-\delta)^2}\right) \cdot f^{2\log n} F_w(S^*) \leq \left(1 + \tfrac{8{c_1}^2\delta}{(1-\delta)^2}\right)(1+\delta) F_w(S^*) \\
&= \left(1 + \delta + \tfrac{8{c_1}^2\delta(1+\delta)}{(1-\delta)^2}\right) F_w(S^*) \leq \left(1 + \delta + \tfrac{16{c_1}^2\delta}{(1-\delta)^2}\right) F_w(S^*) \\
&\leq \left(1 + \tfrac{(16{c_1}^2+1)\delta}{(1-\delta)^2}\right) F_w(S^*) = (1+\epsilon) F_w(S^*)
\end{aligned}$$

Theorem 1. *There is a PTAS for* $P_{1,m}, h_{k_1}, h_{k_2}, \cdots, h_{k_m} \mid nr - a, w_i = p_i \mid \sum w_i C_i$, *its running time is* $O(n^{\frac{c}{\delta}} n(c+m)(\log_f P)^{2(c+m+1)})$, *where* ϵ *is the relative error ratio,* δ *is a constant such that* $\frac{(16c^2+1)\delta}{(1-\delta)^2} \leq \epsilon$, $f = 1 + \frac{\delta}{4\log n}$, $P = \sum p_i$ *and* $c = h_{k_1} + h_{k_2} + \cdots + h_{k_m}$.

Proof. Let $c_1 \leq c$ be the number of bounded intervals and $c_2 \leq m+1$ be the number of unbounded intervals. By Lemma 1, there are at most $O(n^{\frac{c_1}{\delta}})$ possible assignments of the large jobs. For each allocation of large jobs, by Lemma 3, Phase II takes $O(n(c_1+c_2+1)(\log_f P)^{2(c_1+c_2+1)})$, and Phase III and Phase IV are dominated by Phase II. So the total computational time of our algorithm is $O(n^{\frac{c_1}{\delta}} n(c_1+c_2+1)(\log_f P)^{2(c_1+c_2+1)}) = O(n^{\frac{c}{\delta}} n(c+m)(\log_f P)^{2(c+m+1)})$.

2.2 No FTPAS

In this section, we show that the scheduling problem does not admit FPTAS unless $P = NP$. We reduce from the Equal Cardinality Partition (EPC) problem. See Appendix for the reduction.

Theorem 2. *The scheduling problem* $P_{1,2}, 1, 1 \mid nr - a, w_i = p_i \mid \sum w_i C_i$ *does not have FPTAS unless* $P = NP$.

Similarly, we can show that there is no FPTAS for the case of two machines, in which one of them is always available, and the other has two unavailable intervals.

Theorem 3. *The scheduling problem* $P_{1,1}, 2|nr - a, w_i = p_i| \sum w_i C_i$ *does not have FPTAS unless* $P = NP$.

3 Fixed Job Model

In this section, we study the fixed job model. First we show that if the jobs have arbitrary weight, there does not exist a constant approximation algorithm for a single machine with 2 fixed jobs or for two machines with one fixed job on each machine, unless $P = NP$.

Theorem 4. *For any constant* $\alpha > 1$, *there is no polynomial time algorithm that finds an* α*-approximation for* $1, h_2 \mid nr - a, fixed \mid \sum w_i c_i$ *or* $P_{0,2}, h_1, h_1 \mid nr - a, fixed \mid \sum w_i c_i$, *unless* $P = NP$.

Next we study the scheduling problem with $m \geq 1$ identical machines. Like the preventive model, we assume that the number of unavailable intervals due to fixed jobs is also a constant, and the weight of each job is equal to its processing time. Unlike the preventive model, we do not require a machine permanently available.

Theorem 5. *There is a PTAS for* $P_{0,m}, h_{k_1}, h_{k_2}, \cdots, h_{k_m} \mid nr - a,$ *fixed* $, w_i = p_i \mid \sum w_i C_i$, *its running time is* $O(n^{\frac{c}{\delta}} n(c+m)(\log_f P)^{2(c+m)})$, *where* ϵ *is the relative error ratio,* δ *is a constant such that* $\frac{4\delta}{(1-c\delta)^2} \leq \epsilon$, $f = 1 + \frac{\delta}{4 \log n}$, $P = \sum p_i$ *and* c *is the number of fixed jobs.*

References

1. Adiri, I., Bruno, J., Frostig, E., Rinnooy Kan, A.H.G.: Single machine Flow-Time Scheduling with a Single Breakdown. Acta Informatica 26, 679–696 (1989)
2. Baewicz, J., Drozdowski, M., Formanowicz, P., Kubiak, W., Schmidt, G.: Scheduling preemptable tasks on parallel processors with limited availability. Parallel Computing 26(9), 1195–1211 (2000)
3. Cieliebak, M., Eidenbenz, S., Pagourtzis, A., Schlude, K.: Equal Sum Subsets: Complexity of Variations. Technical Report 370, CS, ETHZ (2002)

4. Diedrich, F., Jansen, K.: Improved Approximation Algorithms for Scheduling with Fixed Jobs. In: Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 675–684 (2009)
5. Fu, B., Huo, Y., Zhao, H.: Makespan Minimization with Machine Availability Constraints. Discrete Mathematics, Algorithms and Applications (to appear, 2009)
6. Fu, B., Huo, Y., Zhao, H.: Exponential Inapproximability and FPTAS for Scheduling with Availability Constraints. Theoretical Computer Science 410, 2663–2674 (2009)
7. Kacem, I., Mahjoub, R.: Fully polynomial time approximation scheme for the weighted flow-time minimization on a single machine with a fixed non-availability interval. Computers & Industrial Engineering 56(4), 1708–1712
8. Kaspi, M., Montreuil, B. (1988). On the Scheduling of Identical Parallel Processes with Arbitrary Initial Processor Available TImes, Reserach Report 88-12, School of Industrial Engineering, Purdue University (1988)
9. Kellerer, H., Strusevish, V.A.: Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. Algorithmica (to appear)
10. Kubiak, W., Blazewicz, J., Formanowicz, P., Breit, J., Schmidt, G.: Two-machine flow shops with limited machine availability. European Journal of Operational Research 136, 528–540 (2002)
11. Kubzin, M.A., Potts, C.N., Strusevich, V.A.: Approximation results for flow shop scheduling problems with machine availability constraints. Computers & Operations Research 36(2), 379–390 (2009)
12. Lee, C.Y.: Machine scheduling with availability constraint. Journal of global optimization 9, 395–416 (1996)
13. Lee, C.Y., Liman, S.D.: Single Machine Flow-Time Scheduling With Scheduled Maintenence. Acta Informatica 29, 375–382 (1992)
14. Lee, C.Y.: Machine scheduling with availability constraints. In: Leung, J.Y.-T. (ed.) Handboof of Scheduing, pp. 22.1–22.13. CRC Press, Boca Raton (2004)
15. Liao, L.-W., Sheen, G.-J.: Parallel machine scheduling with machine availability and eligibility constraints. European Journal of Operational Research 184(2), 458–467 (2008)
16. Liman, S.: Scheduling with Capacities and Due-Dates, Ph.D. Dissertation, Industrial and Systems Engineering Department, University of Florida (1991)
17. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Prentice Hall, Englewook Cliffs (1995)
18. Saidy, H., Taghvi-Fard, M.: Study of Scheduling Problems with Machine Availability Constraint. Journal of Industrial and Systems Engineering 1(4), 360–383 (2008)
19. Scharbrodt, M., Steger, A., Weisser, H.: Approximation of scheduling with fixed jobs. Journal of Scheduling 2, 267–284 (1999)
20. Schmidt, G.: Scheduling with limited machine availability. European Journal of Operational Research 121, 1–15 (2000)

An $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ Space Lower Bound for Finding ε-Approximate Quantiles in a Data Stream

Regant Y.S. Hung and Hingfung F. Ting*

The University of Hong Kong, Pofulam, Hong Kong
{yshung,hfting}@cs.hku.hk

Abstract. This paper studies the space complexity of the ε-approximate quantiles problem, which asks for some data structure that enables us to determine, after reading a whole data stream, a ϕ-quantile (for any $0 \leq \phi \leq 1$) of the stream within some error bound ε. The best known algorithm for the problem uses $O(\frac{1}{\varepsilon}\log \varepsilon N)$ words where N is the total number of items in the stream, or uses $O(\frac{1}{\varepsilon}\log |U|)$ words where U is the set of possible items. It is known that the space lower bound of the problem is $\Omega(\frac{1}{\varepsilon})$ words; however, improvement of this bound is elusive.

In this paper, we prove that any comparison-based algorithm for finding ε-approximate quantiles needs $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ words.

1 Introduction

This paper studies the space complexity of finding ε-approximate quantiles in a stream of data items, which can only be read in one pass and any item that is read but is not stored in the memory will be lost forever. All items are from a totally-ordered set U. Given a stream of N items, the ϕ-quantile of this stream is the item whose rank in this stream equals $\lceil \phi N \rceil$. The ε-approximate quantiles problem asks for some data structure such that after reading a data stream of N items, the data structure enables us to find, for any $0 \leq \phi \leq 1$, an item x whose rank is in $[\phi N - \varepsilon N, \phi N + \varepsilon N]$. We call x an ε-approximate ϕ-quantile of the data stream. The main challenge is to use as few memory words as possible. This problem and its variants have been studied extensively [2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 15, 19, 20, 22]. Currently, the best algorithms for the problem are given by Greenwald and Khanna [7], which uses $O(\frac{1}{\varepsilon}\log \varepsilon N)$ words, and by Shrivastava *et al.*[19], which uses $O(\frac{1}{\varepsilon}\log |U|)$ words. It is known that $\Omega(\frac{1}{\varepsilon})$ words are necessary for solving the problem; however, improvement of this lower bound is elusive. In view of this lack of progress in improving the $\Omega(\frac{1}{\varepsilon})$ lower bound, it is natural to ask whether the log factors in the current $O(\frac{1}{\varepsilon}\min\{\log \varepsilon N, \log |U|\})$ upper bound are inherent. We note that for another closely related problem on data streams, namely the ε-approximate frequent items problem, we have $O(\frac{1}{\varepsilon})$-word algorithm [5, 10, 16]. In the IITK Workshop on Algorithms for Data Streams

* This research was supported in part by Hong Kong RGC Grant HKU-7163/07E.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 89–100, 2010.

[1], Cormode raised an open problem of finding the optimal space bound for the ε-approximate quantiles problem. In particular, he asked if $O(\frac{1}{\varepsilon})$-word space is achievable.

This paper gives the first appreciable step for resolving Cormode's open problem. We prove that for a large class of algorithms, namely the class of comparison-based algorithms, we need $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ words to solve the ε-approximate quantiles problem. Furthermore, by some simple extension of our argument, we show that finding ε-approximate ϕ-quantile for one fixed value of ϕ is still difficult. In particular, we prove that any comparison-based algorithm for finding ε-approximate median (i.e., for $\phi = \frac{1}{2}$) needs $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ words. We note that in the literature, there are many comparison-based algorithms for the ε-approximate quantiles problem (e.g.,[2, 3, 4, 7, 12, 14, 22]). In particular, for the two best existing algorithms mentioned above, the $O(\frac{1}{\varepsilon}\log \varepsilon N)$-space algorithm of Greenwald and Khanna [7] is comparison-based, while the $O(\frac{1}{\varepsilon}\log |U|)$-space algorithm of Shrivastava *et al.*[19] is not.

Related Works. Yao[21] is among the first to study the ε-approximate quantiles problem. She proved that any comparison-based algorithm for finding ε-approximate median requires $\Omega(\sqrt{N})$ comparisons. There are also many interesting results on the space complexity, but only for selecting the exact ϕ-quantile (not ε-approximate ϕ-quantile). For example, Pohl [18] showed that if the input can only be read in one pass, any comparison-based algorithm for computing the exact median in a data stream of N items needs to store $N/2$ items. This result was generalized by Munro and Paterson [17], who considered the case when we were allowed to read the data stream in multi-passes. They proved that any comparison-based algorithm for selecting ϕ-quantile in a stream of N items needs $\Omega(N^{\frac{1}{p}}(\log N)^{2-\frac{2}{p}})$ space where p is the number of passes that the algorithm scans the input.

Organization. The paper is organized as follows. In Section 2, we describe formally the computational model that we use in deriving the lower bound, and then give the definitions that are necessary for our discussion. In Section 3, we show how to construct a set of difficult inputs for any algorithm A such that there is always an input that A would return an incorrect answer if it does not have $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ space. Section 4 proves some important properties about this input set, and Section 5 derives the space lower bounds based on these properties.

2 Preliminaries

A data stream is a stream of items that can be read in one pass and any item that has been read but is not stored in the memory will be lost forever. We assume that the set of all possible items are totally ordered. In the rest of this paper, we let M denote the size of the memory. To clarify the discussion, we divide the memory into two parts, the *item memory*, which can store $m \leq M$ items coming from the stream, and the *general memory*, which stores other general values such as pointers, counters, and other system information such as program

counter, registers, etc. A *memory state*, or simply a state, is a pair (I, G) where I is an array of items specifying the content of the item memory (i.e., $I[u]$ is the item stored at the uth location of the memory) and G is an array of values specifying the content of the general memory. We assume that the memory is in the initial state (I_o, G_o) in which for every $1 \leq k \leq m$, $I_o[k]$ stores the null item, which is smaller than all other items, and every $G_o[k]$ contains the value zero. Consider any algorithm A. For any state (I, G) and any item a, we write $A((I, G), a) = (I', G')$ to denote the fact that A updates its state from (I, G) to (I', G') after processing item a, and for any stream $\sigma = a_1 a_2 \ldots a_{i-1} a_i$, the final state of A after processing σ from the state (I, G) can be expressed recursively as $A((I, G), \sigma) = A((I, G), a_1 a_2 \cdots a_{i-1} a_i) = A(A((I, G), a_1 a_2 \cdots a_{i-1}), a_i)$. We say that A is a *comparison-based* algorithm if it can be specified by a decision-tree, which is a ternary tree in which every internal node is labeled by some comparison "xth item : yth item", which compares the items currently stored at the xth and yth location of the item memory (for simplicity, we assume that the item currently being processed is stored at the 0th location). The tree may also have internal nodes labeled by comparisons involving values stored in the general memory or constants that are not items. Every leaf of the tree is labeled with a sequence of operations for updating the current state (I, G) to a new state (I', G'). We assume that the operations for updating I to I' are of one of the following forms:

1. Swap the items stored at the xth and the yth location of the item memory.
2. Replace the item stored at the xth location of the item memory by the one stored at the yth location.

The execution of $A((I, G), a)$ follows a path from the root to a leaf: starting from the root, the comparison labeled at the current internal node will be made and the execution will branch to its left, middle, and right son if the result is $<$, $=$ and $>$, respectively. When the execution reaches a leaf, the sequence of operations labeled at that leaf will be executed to update the state.

Given any two pairs of items (a, a') and (b, b'), we say that the two pairs have the *same relative order* if either $a = a'$ and $b = b'$, or $a < a'$ and $b < b'$, or $a > a'$ and $b > b'$. We say that two states (I_1, G_1) and (I_2, G_2) are *equivalent* if

- for any $1 \leq j, j' \leq m$, $(I_1[j], I_1[j'])$ and $(I_2[j], I_2[j'])$ have the same relative order, and
- $G_1[j] = G_2[j]$ for all j.

Note that if (I_1, G_1) and (I_2, G_2) are equivalent, then there is a permutation $j_1, j_2, \ldots, j_m$ of $1, 2, \ldots m$ such that $I_1[j_1] \leq I_1[j_2] \leq \cdots \leq I_1[j_m]$ and $I_2[j_1] \leq I_2[j_2] \leq \cdots \leq I_2[j_m]$. The following lemma gives some useful properties about comparison-based algorithms.

Lemma 1. *Suppose that the states (I_1, G_1) and (I_2, G_2) are equivalent. Consider any two items a_1 and a_2 such that for all $1 \leq j \leq m$, $(I_1[j], a_1)$ and $(I_2[j], a_2)$ have the same relative order. Then, for any comparison-based algorithm A we have*

(i) the states $(\hat{I}_1, \hat{G}_1) = A((I_1, G_1), a_1)$ *and* $(\hat{I}_2, \hat{G}_2) = A((I_2, G_2), a_2)$ *are equivalent,*
(ii) for any $1 \leq j \leq m$, $\hat{I}_1[j] = a_1$ *if and only if* $\hat{I}_2[j] = a_2$, *and*
(iii) for any $1 \leq i, j \leq m$, $\hat{I}_1[i] = I_1[j]$ *if and only if* $\hat{I}_2[i] = I_2[j]$.

Proof. Since (I_1, G_1) and (I_2, G_2) are equivalent, we have G_1 and G_2 are identical, and there is a permutation $j_1, j_2, \ldots, j_m$ of $1, 2, \ldots, m$ such that $I_1[j_1] \leq I_1[j_2] \leq \cdots \leq I_1[j_m]$ and $I_2[j_1] \leq I_2[j_2] \leq \cdots \leq I_2[j_m]$. Together with the fact that for all $1 \leq j \leq m$, $(I_1[j], a_1)$ and $(I_2[j], a_2)$ have the same relative order, we have

$$I_1[j_1] \leq I_1[j_2] \leq \ldots \leq I_1[j_k] \leq a_1 \leq I_1[j_{k+1}] \leq \cdots \leq I_1[j_m]$$

and

$$I_2[j_1] \leq I_2[j_2] \leq \ldots \leq I_2[j_k] \leq a_2 \leq I_2[j_{k+1}] \leq \cdots \leq I_2[j_m]$$

for some k. It follows that the execution of $A((I_1,G_1),a_1)$ and that of $A((I_2, G_2), a_2)$ follow exactly the same path in the decision tree, and hence will update the item-memory in exactly the same way. The lemma follows.

Below, we give the essential property that a set of streams needs in order to "fool" the comparison-based algorithm A.

Definition 1. *Given any two streams* $\sigma = a_1 a_2 \ldots a_L$ *and* $\pi = b_1 b_2 \ldots b_L$ *of the same length* L, *we say that* σ *and* π *are* indistinguishable *for* A *if after processing* σ *and* π,

1. *the final states* $(I_\sigma, G_\sigma) = A((I_o, G_o), \sigma))$ *and* $(I_\pi, G_\pi) = A((I_o, G_o), \pi)$ *are equivalent, and*
2. *for* $1 \leq j \leq m$, $I_\sigma[j] = a_k$ *if and only if* $I_\pi[j] = b_k$.

Note that for σ and π to be indistinguishable, we only require their items in the item memory to have the same order; the ordering of the other items (i.e., the items that are not in the item memory) may be very different. This is important to our lower bound proofs because it gives us enough flexibility to construct a set of streams in which there is one with large "gap".

Lemma 2. *Suppose that the streams* σ *and* π *are indistinguishable for* A. *Let* $(I_\sigma, G_\sigma) = A((I_0, G_0), \sigma)$ *and* $(I_\pi, G_\pi) = A((I_0, G_0), \pi)$. *Consider any two items* a *and* b *such that for all* $1 \leq j \leq m$, $(I_\sigma[j], a)$ *and* $(I_\pi[j], b)$ *have the same relative order. Then, the streams* σa *and* πb *are indistinguishable for* A.

Proof. Follow directly from Lemma 1.

3 The Construction of Indistinguishable Streams

Suppose that the comparison-based algorithm A has a memory of size $M \leq \frac{1}{12\varepsilon} \ln \frac{1}{12\varepsilon}$. Thus, it can store $m \leq M \leq \frac{1}{12\varepsilon} \ln \frac{1}{12\varepsilon}$ items. In this section, we describe how to construct a set Ψ of difficult input streams for A. We also prove

some simple properties on Ψ. In the next two sections, we give some deeper analysis on the structure of Ψ and then conclude the $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ space lower bound by showing that there exists a stream in Ψ that fools A.

To get Ψ, we iteratively construct the sets of streams $\Psi_0, \Psi_1, \ldots, \Psi_{m-1} = \Psi$. The set Ψ_0 contains only one single stream of $n = 15m$ distinct items, and for $1 \leq i \leq m-1$, the streams in Ψ_i are obtained by duplicating and extending the streams in Ψ_{i-1}. More specifically, for every stream $\sigma \in \Psi_{i-1}$, we create (according to some rules given below) $n-1$ different subsequences $\sigma_{i,1}, \sigma_{i,2}, \ldots, \sigma_{i,n-1}$, each comprises n distinct items. Then, for each $1 \leq k \leq n-1$, we append $\sigma_{i,k}$ to σ and the resulting sequence $\sigma\sigma_{i,k}$ is added to Ψ_i. It can be verified that any stream $\sigma \in \Psi_i$ is the concantenation of some $i+1$ subsequences $\sigma_0, \sigma_1, \ldots, \sigma_i$ where σ_k $(1 \leq k \leq i)$ is appended to the stream during the construction of Ψ_k. For ease of reference, we call σ_k the kth *block* of the stream σ. It is also worth emphasizing that for every $0 \leq i \leq m-1$, the items from the streams in Ψ_i are all distinct.

To detail the construction of Ψ_i from Ψ_{i-1}, we need some defintions and notations. Consider any stream $\sigma = \sigma_0\sigma_1\ldots\sigma_{i-1}$ in Ψ_{i-1} with blocks $\sigma_0, \sigma_1, \ldots, \sigma_{i-1}$.

- We let $\sigma_{p..q}$ denote the sub-sequence $\sigma_p\sigma_{p+1}\ldots\sigma_q$. Note that $\sigma = \sigma_{0..i-1}$. For any item in $\sigma_{0..i-1}$, if it is the ℓth item in block σ_p, we denote it as $\sigma_p[\ell]$. For example, if $\sigma_{0..1} = a_1a_2a_3\ldots a_nb_1b_2\ldots b_n$, then $\sigma_0[3] = a_3$.
- Let $I_A(\sigma)$ be the set of items stored by A after it processes the data stream σ; in other words, if $(I, G) = A((I_o, G_o), \sigma)$, then $I_A(\sigma)$ and I contain the same set of items. For any item $\sigma_p[\ell]$ in σ, we say that $\sigma_p[\ell]$ is *marked* in σ if it is in $I_A(\sigma)$; otherwise, it is *unmarked*.
- Given any two items a, a', and item x, we say that x *sits between* a and a' if either $a < x < a'$ or $a' < x < a$. Given any set of items Z, we say that x *sits within* Z if Z has two items a and a' such that x sits between a and a'.
- Let $\texttt{sort}(\sigma)$ denote the sequence obtained by sorting the items of σ in ascending order.

We are now ready to describe our construction of $\Psi_0, \Psi_1, \ldots, \Psi_{m-1}$. Our construction maintains the following invariants:

In-block-sorted. For each $\sigma \in \Psi_i$ and each block $\sigma_p = \sigma_p[1]\sigma_p[2]\ldots\sigma_p[n]$ of σ, we have $\sigma_p[1] < \sigma_p[2] < \cdots < \sigma_p[n]$.

Indistinguishable for *A*. For any two streams $\sigma, \sigma' \in \Psi_i$, σ and σ' are indistinguishable for A.

Since Ψ_0 has only one sequence, it is easy to construct Ψ_0 that satisfies the above two invariants. Assume that Ψ_{i-1} is in-block-sorted and indistinguishable for A. Note that because of the indistingushable property, all streams in Ψ_{i-1} have the same number of marked items. We now show how to construct Ψ_i from Ψ_{i-1} such that Ψ_i also satisfies the two invariants.

Consider any stream σ in Ψ_{i-1}. Note that the sorted version $\texttt{sort}(\sigma)$ of σ is "chopped-up" by the marked items (i.e., the items in $I_A(\sigma)$) into segments of

unmarked items. We let $gap(\texttt{sort}(\sigma))$ denote the longest segment of unmarked items in $\texttt{sort}(\sigma)$. Define

$$gap(\Psi_{i-1}) = \max\{gap(\texttt{sort}(\sigma)) \mid \sigma \in \Psi_{i-1}\}$$

to be the longest segment of unmarked items among all streams in Ψ_{i-1}. Suppose that $gap(\Psi_{i-1}) = gap(\texttt{sort}(\hat{\sigma}_{0..i-1}))$ for some stream $\hat{\sigma}_{0..i-1} \in \Psi_{i-1}$. By definition, $gap(\texttt{sort}(\hat{\sigma}_{0..i-1}))$ is enclosed by two marked items, say, $\hat{\sigma}_p[\ell]$ and $\hat{\sigma}_q[\ell']$ in $\texttt{sort}(\hat{\sigma}_{0..i-1})$.[1] Suppose that $\hat{\sigma}_p[\ell]$ is the $\hat{r}$th marked item (from the left) in $\texttt{sort}(\hat{\sigma}_{0..i-1})$. Then, $\hat{\sigma}_q[\ell']$ is the $(\hat{r}+1)$st marked item in $\texttt{sort}(\hat{\sigma}_{0..i-1})$. Our construction of Ψ_i from Ψ_{i-1} is based on this value of $\hat{r}$.

> For every stream $\sigma_{0..i-1} \in \Psi_{i-1}$, we create $n-1$ blocks as follows. Let x and y be the $\hat{r}$th and $(\hat{r}+1)$st marked items in $\texttt{sort}(\sigma_{0..i-1})$. Let $g_1 g_2 \ldots g_L$ be the segment of items between x and y in $\texttt{sort}(\sigma_{0..i-1})$. For $1 \le k \le n-1$, we create a block $\sigma_{i,k} = b_1^k b_2^k \ldots b_n^k$ of n new and distinct items where
>
> $$x < b_1^k < b_2^k < \cdots < b_k^k < g_1 < \cdots < g_L < b_{k+1}^k < \cdots < b_n^k < y. \quad (\dagger)$$
>
> Then, for $1 \le k \le n-1$, we append $\sigma_{i,k}$ to $\sigma_{0..i-1}$ and the resulting stream $\sigma_{0..i-1}\sigma_{i,k}$ will be put into Ψ_i.[2] For ease of reference, we call x and y respectively the *left-anchor* and *right-anchor* of $\sigma_{i,k}$.

It is obvious that Ψ_i is in-block-sorted. The following lemma proves that it is also indistinguishable.

Lemma 3. *The set Ψ_i is indistinguishable for A.*

Proof. Consider any two streams $\sigma_{0..i-1}\sigma_i$ and $\pi_{0..i-1}\pi_i$ in Ψ_i. Since $\sigma_{0..i-1}$ and $\pi_{0..i-1}$ are in Ψ_{i-1}, they are indistinguishable for A. Thus, if $(I_\sigma, G_\sigma) = A((I_o, G_o), \sigma_{0..i-1})$ and $(I_\pi, G_\pi) = A((I_o, G_o), \pi_{0..i-1})$, then $(I_\sigma[u], I_\sigma[v])$ and $(I_\pi[u], I_\pi[v])$ have the same relative order for any $1 \le u, v \le m$; in other words, if $I_\sigma[u_1] \le I_\sigma[u_2] \le \cdots \le I_\sigma[u_m]$ are the sorted sequence of the items in I_σ, then $I_\pi[u_1] \le I_\pi[u_2] \le \cdots \le I_\pi[u_m]$ are the sorted sequence of items in I_π. We can rewrite $(\dagger)$ for the creation of σ_i and π_i as follows:

$$I_\sigma[u_{\hat{r}}] < \sigma_i[1] < \cdots < g_1 \ldots g_L \ < \cdots < \sigma_i[n] < I_\sigma[u_{\hat{r}+1}]$$
$$I_\pi[u_{\hat{r}}] < \pi_i[1] < \cdots < g'_1 \ldots g'_{L'} < \cdots < \pi_i[n] < I_\pi[u_{\hat{r}+1}].$$

Observe that $(I_\sigma[u], \sigma_i[1])$ and $(I_\pi[u], \pi_i[1])$ have the same relative order for all $1 \le u \le m$. By Lemma 2, we conclude that $\sigma_{0..i-1}\sigma_i[1]$ and $\pi_{0..i-1}\pi_i[1]$ are indistinguishable. By applying Lemma 2 repeatedly to $(\sigma_i[2], \pi_i[2]), \ldots, (\sigma_i[n], \pi_i[n])$, we conclude that $\sigma_{0..i-1}\sigma_i$ and $\pi_{0..i-1}\pi_i$ are indistinguishable. The lemma follows.

[1] To handle the exception case when the segment starts from the first item or ends at the last item, we may assume that there are two additional marked items enclosing $\texttt{sort}(\hat{\sigma}_{0..i-1})$.

[2] Note that if there is no items between x and y in $\texttt{sort}(\sigma_{0..i-1})$, then we will only create one $\sigma_{0..i-1}\sigma_i$.

4 A Lower Bound on the Length of $gap(\Psi_{m-1})$

In this section, we prove that there is a stream $\sigma \in \Psi_{m-1}$ such that the length $|gap(\sigma)|$ of $gap(\sigma)$ is at least $\frac{14}{4\ln 4} m \log m$. Note that σ has length $N = nm \approx (\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})^2$ and thus $|gap(\sigma)| \geq \frac{14}{4\ln 4} m \log m \approx \frac{1}{\varepsilon}(\log\frac{1}{\varepsilon})^2 = \Theta(\varepsilon N)$, which is (asymptotically) the bound we target for fooling A.

Let $\sigma = \sigma_0\sigma_1 \ldots \sigma_{m-1}$ be any stream in Ψ_{m-1}. Consider any block σ_j of σ. By the in-block-sorted property, $\texttt{sort}(\sigma_j)$ is identical to σ_j, and by our construction, σ_j may be cut into two segments in $\texttt{sort}(\sigma_{j..j+1})$ after appending σ_{j+1} to σ_j. It may be cut into even more smaller segments after appending more blocks later. The following definition is central to the discussion in this section.

> For any $0 \leq j \leq i \leq m-1$, we say that a sequence of items is a *j-segment in* $\texttt{sort}(\sigma_{j..i})$ if it is a segment (i.e., a sequence of consecutive items) in $\texttt{sort}(\sigma_{j..i})$ containing only items in the block σ_j.

We say that a segment *avoids* the item set Z if the segment contains no items in Z. Note that if there is a segment in $\texttt{sort}(\sigma_{0..i})$ that avoids $I_A(\sigma_{0..i})$ and has length ℓ, then $|gap(\sigma_{0..i})| \geq \ell$. The lemma below suggests that if there is a j-segment in $\texttt{sort}(\sigma_{j..i})$ that avoids $I_A(\sigma_{0..i})$ and has length ℓ, then we can also conclude that $|gap(\sigma_{0..i})| \geq \ell$.

Lemma 4. *Suppose that* $\texttt{sort}(\sigma_{0..i}) = a_1 a_2 \ldots a_{|\sigma_{0..i}|}$. *Consider any j-segment* $H = a_{i_1} a_{i_2} \ldots a_{i_k}$ *in* $\texttt{sort}(\sigma_{j..i})$.

1. *For any item $a \in \sigma_{0..j-1}$ that sits within H, $a \notin I_A(\sigma_{0..\ell})$ for all $j-1 \leq \ell \leq i$.*
2. *If $H = a_{i_1} a_{i_2} \ldots a_{i_k}$ avoids $I_A(\sigma_{0..i})$, then the corresponding complete segment $\hat{H} = a_{i_1} a_{i_1+1} \ldots a_{i_2} \ldots\ a_{i_k-1} a_{i_k}$ in $\texttt{sort}(\sigma_{0..i})$ also avoids $I_A(\sigma_{0..i})$.*

Proof. For (1), recall that the items of σ_j are sitting between two consecutive marked items in $\texttt{sort}(\sigma_{0..j-1})$, and as a sits within σ_j, it is not marked and thus is not in $I_A(\sigma_{0..j-1})$). Together with the fact that a will not reappear, A will not see or remember a after processing $\sigma_{0..j-1}$; this implies $a \notin I_A(\sigma_{0..\ell})$ for $j-1 \leq \ell \leq i$. For (2), note that for any $a \in \hat{H} - H$, $a \notin \sigma_{j+1..i}$ because H is a j-segment in $\texttt{sort}(\sigma_{j..i})$. Thus, a is from $\sigma_{0..j-1}$ and by (1), it is not in $I_A(\sigma_{0..i})$. Together with the fact that H avoids $I_A(\sigma_{0..i})$, the statement follows.

Our plan is to show that there exists a long j-segment. To this end, we need to study how σ_j is cut by later segments. To be precise, for any $j < k \leq m-1$, we say that σ_j *is cut* by σ_k if we have inserted some new items within a j-segment in $\texttt{sort}(\sigma_{j..k-1})$ when creating σ_k. According to our construction, we have the following fact.

Fact 1. *If σ_j is cut by σ_k, then σ_k's left-anchor or right-anchor or both are items of σ_j.*

If both σ_k's left- and right-anchors are items of σ_j, we say that σ_k cuts σ_j twice; otherwise, we say that σ_k cuts σ_j once. We now study how many segments can

be obtained by cutting σ_j. For any $0 \le j \le i$, let n_j be the number of maximal j-segments in $\texttt{sort}(\sigma_{j..i})$ that avoid $I_A(\sigma_{0..i})$. The following lemma derives an upper bound on their sum. (Note that the sum does not include n_0; our later analysis does not consider n_0.)

Lemma 5. *We have* $\sum_{1\le j\le i} n_j \le 3i + m$ *and thus there exists a* j *with* $n_j \le (3i + m)/i$.

Proof. The inequality can be derived by a simple double counting argument. Let T be a table with $i + |I_A(\sigma_{0..i})|$ rows and i columns. Each of the first i rows of T are labeled by a distinct block σ_k for $1 \le k \le i$ and each of the remaining $|I_A(\sigma_{0..i})|$ rows are labeled by a distinct item in $I_A(\sigma_{0..i})$. Each column of T is labeled by a distinct σ_j for $1 \le j \le i$. For any row σ_k and column σ_j, we let

$$T[\sigma_k, \sigma_j] = \begin{cases} 2 & \text{if } \sigma_k \text{ cuts } \sigma_j \text{ twice;} \\ 1 & \text{if } \sigma_k \text{ cuts } \sigma_j \text{ once;} \\ 0 & \text{otherwise,} \end{cases}$$

and for any row $a \in I_A(\sigma_{0..i})$, we set $T[a, \sigma_j]$ to 1 if $a \in \sigma_j$, and set it to 0 otherwise. Observe that starting from a single j-segment, we may increase the number of maximal j-segment by 1 when we find that σ_j is cut by a block, or when we find an item of σ_j in $I_A(\sigma_{0..i})$ (recall that we are interested in maximal j-segments that avoid $I_A(\sigma_{0..i})$). It follows that

$$n_j \le \textstyle\sum_{1\le k\le i} T[\sigma_k, \sigma_j] + \sum_{a\in I_A(\sigma_{0..i})} T[a, \sigma_j] + 1. \tag{1}$$

Fact 1 implies that for any $1 \le k \le i$, the sum of all entries of row σ_k is at most 2. For any row $a \in I_A(\sigma_{0..i})$, since the items in $\sigma_0, \sigma_1, \ldots, \sigma_i$ are all distinct, a can be in one σ_j; in other words, there is a 1 in row a. Therefore, $\sum_{1\le j\le i}\left(\sum_{1\le k\le i} T[\sigma_k, \sigma_j] + \sum_{a\in I_A(\sigma_{0..i})} T[a, \sigma_j]\right) = \left(\sum_{1\le k\le i}\sum_{1\le j\le i} T[\sigma_k, \sigma_j]\right) + \left(\sum_{a\in I_A(\sigma_{0..i})}\sum_{1\le j\le i} T[a, \sigma_j]\right) \le 2i + |I_A(\sigma_{0..i})| \le 2i + m$. Together with (1), the lemma follows.

When $i = m-1$, Lemma 5 asserts that there exists some $1 \le j \le m-1$ such that the number of maximal j-segments in $\texttt{sort}(\sigma_{j..m-1})$ that avoids $I_A(\sigma_{0..m-1})$ is at most 4. Since at most m of the n items of σ_j are in $I_A(\sigma_{0..m-1})$, we have at least $n - m = 14m$ items to be distributed into four or fewer segments. It follows that there is one with length at least $14m/4$ and by Lemma 4(2), we have $gap(\Psi) = \Omega(m)$. Unfortunately, this is much smaller than our target $\Omega(m \log m)$ bound. To make improvement, the novel idea is to rewrite the inequality in Lemma 5 as follows.

Lemma 6

1. *We have* $\sum_{1\le j\le i} n_j \le \sum_{1\le j\le i} \frac{(3i+m)\ln 4}{(i+1)\ln(i+1) - j\ln j}$ *and thus there is a* $1 \le j \le i$ *with* $n_j \le \frac{(3i+m)\ln 4}{(i+1)\ln(i+1) - j\ln j}$.

2. *For some* $1 \le j \le i \le m-1$, *there is a* j*-segment in* $\texttt{sort}(\sigma_{j..i})$ *that avoids* $I_A(\sigma_{0..i})$ *and has length at least* $\frac{n-m}{(3i+m)\ln 4}((i+1)\ln(i+1) - j\ln j) > \frac{14m}{4m\ln 4}((i+1)\ln(i+1) - j\ln j)$.

Proof. By Lemma 5, (1) is true if $\sum_{1\le j\le i} \frac{\ln 4}{(i+1)\ln(i+1) - j\ln j} \ge 1$. Note that for any $1 \le b \le a$, $a\ln a - b\ln b = (a-b)\ln a + b\ln(1+\frac{a-b}{b}) \le (a-b)\ln a + (a-b) = (a-b)(1+\ln a)$. Therefore, $\sum_{1\le j\le i} \frac{\ln 4}{(i+1)\ln(i+1) - j\ln j} \ge \sum_{1\le j\le i} \frac{\ln 4}{(i+1-j)(1+\ln(i+1))} = \frac{\ln 4}{1+\ln(i+1)}\left(\frac{1}{i} + \cdots + 1\right) > \frac{\ln 4 \ln i}{1+\ln(i+1)}$, which is greater than 1 for all $i > 40$ because $\frac{\ln 4 \ln 41}{1+\ln(41+1)} > 1.08$ and the function $\frac{\ln x}{1+\ln(x+1)}$ is monotonically increasing. It can be verified that the sum is also greater than 1 for all $i \le 40$. Statement (1) follows.

Statement (2) follows from Statement (1).

By applying Lemma 6(2) repeatedly, we get, for some $j_1 > j_2 > \cdots > 1$, a j_1-segment in $\texttt{sort}(\sigma_{j_1..m-1})$ of length $\Omega(m\log m - j_1\log j_1)$, a j_2-segment in $\texttt{sort}(\sigma_{j_2..j_1-1})$ with length $\Omega(j_1\log j_1 - j_2\log j_2)$ and so on; all these segments avoid $I_A(\sigma_{0..m-1})$ and have total length $\Omega((m\log m - j_1\log j_1) + (j_1\log j_1 - j_2\log j_2) + \ldots) = \Omega(m\log m)$. If the items in these segments form a single segment in $\texttt{sort}(\sigma_{0..m-1})$, then we can conclude that $gap(\Psi_{m-1}) = \Omega(m\log m)$. By making use of the indistinguishable property of the Ψ_i's, we show in the rest of this section that this is true for at least one $\sigma_{0..m-1} \in \Psi_{m-1}$.

Consider any $1 \le j < i \le m-1$. Note that for any two streams in Ψ_j, they are indistinguishable and look the same to A; thus they will be "cut" in exactly the same way during the appending of later blocks $\sigma_{j+1}, \ldots, \sigma_i$. The following lemma proves this observation formally.

Lemma 7. *Consider any two streams* $\sigma_{0..i}$ *and* $\pi_{0..i}$ *in* Ψ_i. *If*

$\sigma_j[u]\sigma_j[u+1]\ldots\sigma_j[v]$ *is a* j*-segment in* $\texttt{sort}(\sigma_{j..i})$ *that avoids* $I_A(\sigma_{0..i})$, *then*
$\pi_j[u]\pi_j[u+1]\ldots\pi_j[v]$ *is a* j*-segment in* $\texttt{sort}(\pi_{j..i})$ *that avoids* $I_A(\pi_{0..i})$.

Proof. Lemma 3 asserts that Ψ_i, and hence $\pi_{0..i}$ and $\sigma_{0..i}$, are indistinguishable for A. Together with the fact that $\sigma_j[u]\sigma_j[u+1]\ldots\sigma_j[v]$ avoids $I_A(\sigma_{0..i})$, we conclude that $H = \pi_j[u]\pi_j[u+1]\ldots\pi_j[v]$ avoids $I_A(\pi_{0..i})$ (see Definition 1). Below, we prove that H is a j-segment in $\texttt{sort}(\pi_{j..i})$.

Suppose to the contrary that H is not a j-segment in $\texttt{sort}(\pi_{j..i})$. Then, there must be a $j+1 \le k \le i$ such that H is a j-segment in $\texttt{sort}(\pi_{j..k-1})$ and not in $\texttt{sort}(\pi_{j..k})$ (note that H is a j-segment in $\texttt{sort}(\pi_{j..j})$). It follows that π_k has some item(s) sitting within H. Referring to (†), note that when constructing π_k, the items of π_k are sitting within $\pi_{0..k-1}$ as follows:

$$x < \pi_k[1] < \cdots < \pi_k[\ell] < g_1 < \cdots < g_L < \pi_k[\ell+1] < \cdots < \pi_k[n] < y$$

where the left-anchor x and right-anchor y are in $I_A(\pi_{0..k-1})$, and hence in $\pi_{0..k-1}$. We prove below that x and y cannot be both items of $\pi_{0..k-1}$, and this leads to contradiction.

There are two possible cases: (1) $\pi_k[1], \ldots, \pi_k[\ell]$ sit within $H = \pi_j[u] \ldots \pi_j[v]$, and (2) $\pi_k[\ell+1], \ldots, \pi_k[n]$ sit within H. For Case (1), we have

$$\pi_j[u] < \cdots < \pi_j[z] \leq x < \pi_k[1] < \cdots < \pi_k[\ell] < \pi_j[z+1] < \cdots < \pi_j[v]$$

for some $u \leq z < v$. Note that

- x is not in $\pi_{j+1..k-1}$ because H is a j-segment in $\texttt{sort}(\pi_{j..k-1})$ and no item from $\pi_{j+1..k-1}$ sits within H, and
- x is not in $\pi_{0..j-1}$ because $x \in I_A(\pi_{0..k-1})$ and no item from $\pi_{0..j-1}$ that sits within H can be in $I_A(\pi_{0..k-1})$ (see Lemma 4(1)).

Therefore, the only remaining possibility is that x is from π_j. This implies $\pi_j[z] = x$. Suppose that $\pi_j[z]$ is the rth marked item in $\texttt{sort}(\pi_{0..k-1})$. Since $\pi_{0..k-1}$ and $\sigma_{0..k-1}$ are indistinguishable by A, $\sigma_j[z]$ is also the rth marked item in $\texttt{sort}(\sigma_{0..k-1})$, and when constructing σ_k, its items will be inserted immediately following $\sigma_j[z]$. Thus, there is some item of σ_k sitting between $\sigma_j[z]$ and $\sigma_j[z+1]$, and $\sigma_j[u]\sigma_j[u+1] \ldots \sigma_j[v]$ is not a j-segment in $\texttt{sort}(\sigma_{j..i})$, a contradiction. Thus, x cannot be an item of π_j either.

Similarly, for Case (2), we can conclude that y is not an item of $\pi_{0..k-1}$. Thus, as claimed, x and y cannot be both items of $\pi_{0..k-1}$.

We are now ready to derive a lower bound on the length of $gap(\Psi_i)$.

Lemma 8. *For any $0 \leq i \leq m-1$, we have $|gap(\Psi_i)| \geq c(i+1)\ln(i+1)$ where $c = \frac{14}{4 \ln 4}$.*

Proof. We prove the lemma by induction. It is obviously true for $i = 0$. Suppose that it is true for $i-1$ and we consider Ψ_i.

By Lemmas 6 and 7, there exist $1 \leq j \leq i$, and $1 \leq u < v \leq n$ such that for any stream $\sigma_{0..i} \in \Psi_i$, $\sigma_j[u]\sigma_j[u+1] \cdots \sigma_j[v]$ is a j-segment in $\texttt{sort}(\sigma_{j..i})$ that avoids $I_A(\sigma_{0..i})$ and its length is at least $c((i+1)\ln(i+1) - j \ln j)$. By the induction hypothesis, $|gap(\Psi_{j-1})| \geq cj \ln j$ and there is a $\hat{\sigma}_{0..j-1} \in \Psi_{j-1}$ such that $gap(\hat{\sigma}_{0..j-1}) = g_1 g_2 \ldots g_L$ has length $|gap(\Psi_{j-1}| \geq cj \ln j$. By construction, we will extend $\hat{\sigma}_{0..j-1}$ into $n-1$ different streams in Ψ_j by appending $n-1$ different $\hat{\sigma}_j$, and one of them will satisfy

$$x < \hat{\sigma}_j[1] < \ldots < \hat{\sigma}_j[u] < g_1 < \cdots < g_L < \hat{\sigma}_j[u+1] < \cdots < \hat{\sigma}_j[v] < \cdots < \hat{\sigma}_j[n] < y.$$

Thus, the sequence $\hat{\sigma}_j[u] g_1 \ldots g_L \hat{\sigma}_j[u+1] \ldots \hat{\sigma}_j[v]$ has length at least $c(i+1)\ln(i+1)$. It follows that the complete segment $\hat{\sigma}_j[u] \cdots \hat{\sigma}_j[v]$ in $\hat{\sigma}_{0..i}$ has length at least $c(i+1)\ln(i+1)$ and by Lemma 4(2), it avoids $I_A(\sigma_{0..i})$. The lemma follows.

5 A Lower Bound on the Space Complexity

We now apply the results in the previous section to derive the lower bound claimed in this paper. Then, we adapt our lower bound proof and derive the same $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ lower bound on the space complexity for finding ε-approximate median in a data stream.

Theorem 2. *Any comparison-based algorithm A with memory of size $m \leq \frac{1}{12\varepsilon}\ln\frac{1}{12\varepsilon}$, cannot solve the ε-approximate quantiles problem.*

Proof. Without loss of generality, we assume that $\ln\frac{1}{12\varepsilon} \geq 1$; otherwise the $\Omega(\frac{1}{\varepsilon})$ lower bound implies the $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ bound. Recall that $n = 15m$ and the length of a stream in $\Psi = \Psi_{m-1}$ is $N = nm$. By Lemma 8, there is a stream $\sigma_{0..m-1} \in \Psi$ with gap length $|gap(\sigma_{0..m-1})| = g \geq \frac{14}{4\ln 4}m\ln m = \frac{N}{4\ln 4}\frac{14}{15}\frac{\ln m}{m} \geq \frac{N}{4\ln 4}\frac{14}{15}\frac{12\varepsilon}{\ln\frac{1}{12\varepsilon}}\ln(\frac{1}{12\varepsilon}\ln\frac{1}{12\varepsilon}) \geq \frac{N}{4\ln 4}\frac{14\cdot 12\varepsilon}{15} > 2\varepsilon N$. Out of the $g \geq 2\varepsilon N + 1$ items in $gap(\sigma_{0..m-1})$, we consider $2\varepsilon N + 1$ consecutive items in the gap. Assume that these items have ranks from $r - \varepsilon N$ to $r + \varepsilon N$ in $\mathtt{sort}(\sigma_{0..m-1})$ for some integer r. Then, to find an ε-approximate ϕ-quantile with $\phi = \frac{r}{N}$, A must return an item with rank in $[\phi N - \varepsilon N, \phi N + \varepsilon N] = [r - \varepsilon N, r + \varepsilon N]$. It is not possible for A to return any of these items because they are all in a gap and are not in A's memory.

Note that to solve ε-approximate quantiles problem, we are required to find ε-approximate ϕ-quantile for any $1 \leq \phi \leq N$. The following theorem suggests that $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ space is still necessary even if we only need to find ε-approximate ϕ-quantile for some fixed ϕ.

Theorem 3. *Any comparison-based algorithm A with memory of size $o(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ cannot find any ε-approximate median (i.e., 0.5-quantile) in a stream.*

Proof. As argued in the proof of Theorem 2, for any comparison-based algorithm A with $o(\frac{1}{\varepsilon_o}\log\frac{1}{\varepsilon_o})$-word space, there is a stream σ of N items such that $\mathtt{sort}(\sigma)$ has a segment of at least $2\varepsilon_o N + 1$ items that avoids $I_A(\sigma)$. By letting $\varepsilon_o = 2\varepsilon$, we conclude that $\mathtt{sort}(\sigma)$ has a segment of at least $4\varepsilon N + 1$ items that avoids $I_A(\sigma)$. Let a and b be the smallest and the largest item among these $4\varepsilon N + 1$ items, respectively. Let r be the rank of a. Note that $1 \leq r \leq N - 4\varepsilon N$. Suppose that N more items arrive such that $N - r - 2\varepsilon N$ of them are smaller than a and the remaining items are larger than b. In the resultant stream $\hat{\sigma}$ of $2N$ items, the ranks of those $4\varepsilon N+1$ items in $\hat{\sigma}$ are $[(N-r-2\varepsilon N)+r, (N-r-2\varepsilon N)+r+4\varepsilon N] = [N - 2\varepsilon N, N + 2\varepsilon N]$, and they avoid $\mathtt{sort}(\hat{\sigma})$; it follows that A cannot return any ε-approximate median for $\hat{\sigma}$.

References

1. http://www.cse.iitk.ac.in/users/sganguly/workshop.html
2. Agrawal, R., Swami, A.: A one-pass space-efficient algorithm for finding quantiles. In: Proceedings of the 7th International Conference on Management of Data, pp. 28–30 (1995)
3. Alsabti, K., Ranka, S., Singh, V.: A one-pass algorithm for accurately estimating quantiles for disk-resident data. In: Proceedings of 23rd International Conference on Very Large Data Bases, pp. 346–355 (1997)
4. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: Proceedings of the 23rd ACM Symposium on Principles of Database Systems, pp. 286–296 (2004)

5. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Proceedings of the 10th Annual European Symposium on Algorithms, pp. 348–360 (2002)
6. Govindaraju, N.K., Raghuvanshi, N., Manocha, D.: Fast and approximate stream mining of quantiles and frequencies using graphics processors. In: Proceedings of the 24th ACM SIGMOD, pp. 611–622 (2005)
7. Greenwald, M., Khanna, S.: Space-efficient online computation of quantile summaries. In: Proceedings of the 20th ACM SIGMOD, pp. 58–66 (2001)
8. Guha, S., McGregor, A.: Approximate quantiles and the order of the stream. In: Proceedings of the 25th ACM Symposium on Principles of Database Systems, pp. 273–279 (2006)
9. Jain, R., Chlamtac, I.: The p^2 algorithm for dynamic calculation for quantiles and histograms without storing observations. Communication of ACM 28, 1076–1085 (1985)
10. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Transactions on Database Systems 28(1), 51–55 (2003)
11. Lin, X.: Continuously maintaining order statistics over data streams: extended abstract. In: Proceedings of the 18th Conference on Australasian Database, pp. 7–10 (2007)
12. Lin, X., Lu, H., Xu, J., Yu, J.X.: Continuously maintaining quantile summaries of the most recent N elements over a data stream. In: Proceedings of the 20th International Conference on Data Engineering, pp. 362–374 (2004)
13. Lin, X., Xu, J., Zhang, Q., Lu, H., Zhou, X., Yuan, Y.: Approximate processing of massive continuous quantile queries over high-speed data streams. IEEE Transactions on Knowledge and Data Engineering (TKDE) 18(5), 683–698 (2006)
14. Manku, G.S., Rajagopalan, S., Lindsay, B.G.: Approximate medians and other quantiles in one pass and with limited memory. In: Proceedings of the 17th ACM SIGMOD, pp. 426–435 (1998)
15. Manku, G.S., Rajagopalan, S., Lindsay, B.G.: Random sampling techniques for space efficient online computation of order statistics of large datasets. In: Proceedings of the 18th ACM SIGMOD, pp. 251–262 (1999)
16. Misra, J., Gries, D.: Finding repeated elements. Science of Computer Programming 2, 143–152 (1982)
17. Munro, J.I., Paterson, M.: Selection and sorting with limited storage. Theoretical Computer Science 12, 315–323 (1980)
18. Pohl, I.: A minimum storage algorithm for computing median. IBM Research Report RC 2701, IBM T.J. Watson Center (1969)
19. Shrivastava, N., Buragohain, C., Agrawal, D., Suri, S.: Medians and beyond: new aggregation techniques for sensor networks. In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, pp. 239–249 (2004)
20. Xu, J., Lin, X., Zhou, X.: Space efficient quantile summary for constrained sliding windows on a data stream. In: Li, Q., Wang, G., Feng, L. (eds.) WAIM 2004. LNCS, vol. 3129, pp. 34–44. Springer, Heidelberg (2004)
21. Yao, F.F.: On lower bounds for selection problems. Technical Report MAC TR-121, Massachusetts Institute of Technology (1974)
22. Zhang, Q., Wang, W.: A fast algorithm for approximate quantiles in high speed data streams. In: Proceedings of the 19th International Conference on Statistical and Scientific Database Management (SSDBM), p. 29 (2007)

Improved Sublinear Time Algorithm for Width-Bounded Separators*

Liang Ding[1], Bin Fu[1], and Yunhui Fu[1]

Department of Computer Science, University of Texas-Pan American, Edinburg, TX 78539, USA
dliang@broncs.utpa.edu, binfu@cs.panam.edu, fuyunhui@gmail.com

Abstract. A width-bounded separator is a simple structured hyperplane which divides the given set into two balanced subsets, while at the same time maintaining a low density of the set within a given distance to the hyperplane. For a given set Q of n grid points in a d-dimensional Euclidean space, we develop an improved (Monte carlo) algorithm to find a w-wide separator L in $\tilde{O}(n^{\frac{1}{d}})$ sublinear time such that Q has at most $(\frac{d}{d+1}+o(1))n$ points on one either side of the hyperplane L, and at most $c_d w n^{\frac{d-1}{d}}$ points within $\frac{w}{2}$ distance to L, where c_d is a constant for fixed d. This improves the existing $\tilde{O}(n^{\frac{2}{d}})$ algorithm by Fu and Chen. Furthermore, we derive an $\Omega(n^{\frac{1}{d}})$ time lower bound for any randomized algorithm that tests if a given hyperplane satisfies the conditions of width-bounded separator. This lower bound almost matches the upper bound.

1 Introduction

Separator is a fundamental tool in algorithm design. In past decades, many efforts focus on generalization of planar graph separators which play critical roles in development of separator theory. The plannar separator theorem, originally due to Lipton and Tarjan [3], states that every n vertex planar graph has at most $\sqrt{8n}$ vertices whose removal separates the graph into two disconnected parts of size at most $\frac{2}{3}n$. Their $\frac{2}{3}$-separator has been improved by a series of papers [4,5,6,7] with the best record $1.97\sqrt{n}$ by Djidjev and Venkatesan [7]. Spielman and Teng [8] showed a $\frac{3}{4}$-separator with size $1.82\sqrt{n}$ for planar graphs. Separators for more general graphs were derived in [9,10,11]. A planar graph can be induced by a set of non-overlapping discs on the plane such that every vertex corresponds to a disc center and each edge corresponds to a tangent relationship between two discs. The separator developed by Miller, Teng and Vavasis [12] is a generalization of planar graph separators to the d-dimensional Euclidean space. Some $O(\sqrt{k \cdot n})$ size separators for k-thick system, in which every point is covered by at most k objects, and the related algorithms were derived in [12,13,14,15].

* This research is supported by NSF Career Award 0845376.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 101–112, 2010.

The study of width-bounded separators were initiated by Fu in [16] and has yielded successful applications in [2,17]. Width-bounded geometric separator has some interesting advantages over previous geometric separators such as the popular geometric separator by Miller, Teng and Vavasis [12]. The width-bounded separator has a simpler linear structure and better balance than that by Miller *et al.*'s separator, which is a sphere and can be found in linear time [18].

In this paper, we give an improved algorithm which finds a width-bounded separator with high probability in $\tilde{O}(n^{\frac{1}{d}})$ sublinear time in a dimensioinal Euclidean space $\Re^d$ for $d \geq 2$. Our algorithm is similar to that developed by Fu and Chen [1], but the new algorithm is more careful to control the point sampling and probability to guarantee the balance and low-density conditions. To our best knowledge, this is the first time to give sublinear time algorithm for finding a $2D$ width-bounded separators. Further more, we derive a $\Omega(n^{\frac{1}{d}})$ lower bound for any randomized algorithm that tests if a given hyperplane satisfies the conditions of width-bounded separator. This lower bound almost matches the upper bound.

2 Preliminaries and Width-Bounded Separators

For any finite set A, $|A|$ denotes the number of elements in A. Let $\Re$ be the set of all real numbers. For two points p_1, p_2 in the d-dimensional Euclidean space $\Re^d$, $\text{dist}(p_1, p_2)$ is the Euclidean distance between p_1 and p_2. For a set $A \subseteq \Re^d$, $\text{dist}(p_1, A) = \min_{q \in A} \text{dist}(p_1, q)$. The *diameter* of any $P \subseteq \Re^d$ is $\max_{p_1, p_2 \in P} \text{dist}(p_1, p_2)$. For $a > 0$ and a set A of points in $\Re^d$, if the distance between every two points in A is at least a, then A is called *a-separated*. For $\epsilon > 0$ and a set Q of points in $\Re^d$, an *ϵ-sketch* of Q is another set P of points in $\Re^d$ such that each point in Q has a distance $\leq \epsilon$ to some point in P. We say P is a sketch of Q if P is an ϵ-sketch of Q for some constant $\epsilon > 0$ (that does not necessarily depend on the size of Q). A sketch set is usually an 1-separated set such as a grid point set. A weight function $w : P \to [0, \infty)$ is often used to measure the density of Q near each point in P. Let $f : \Re^d \to \Re$ be a smooth function. Its *surface* is the set $L(f) = \{v \in \Re^d | f(v) = 0\}$. A *hyperplane* in $\Re^d$ through a fixed point $p_0 \in \Re^d$ is defined by the equation $(p - p_0) \cdot v = 0$, where v is a normal vector of the plane and " ." is the usual vector inner product. A hyperplane in $\Re^d$ is determined by $L(f)$ for some linear function $f : \Re^d \to \Re$. A function $f(n)$ is $\tilde{O}(g(n))$ if $f(n) = O(g(n)(\log n)^c)$ for some constant c.

Definition 1. *Given any $Q \subseteq \Re^d$ with a sketch $P \subseteq \Re^d$, a constant $a > 0$, and a weight function $w : P \to [0, \infty)$, an a-wide-separator is determined by the surface $L(f)$ for some linear function $f : \Re^d \to \Re$. The separator has two measurements for its quality of separation: (1)* $\text{balance}(L(f), Q) = \frac{\max(|Q_1|, |Q_2|)}{|Q|}$, *where* $Q_1 = \{q \in Q | f(q) < 0\}$ *and* $Q_2 = \{q \in Q | f(q) > 0\}$*; and (2)* $\text{density}(L(f), P, \frac{a}{2}, w)$, *where in general* $\text{density}(A, P, x, w) = \sum_{p \in P, \text{dist}(p, A) \leq x} w(p)$ *for any* $A \subseteq \Re^d$ *and* $x > 0$. *When f is fixed or no confusion arises, we use* $\text{balance}(L, Q)$ *and* $\text{density}(L, P, \frac{a}{2}, w)$ *to stand for* $\text{balance}(L(f), Q)$ *and* $\text{density}(L(f), P, \frac{a}{2}, w)$, *respectively.*

Definition 2. *A (b,c)-partition of $\Re^d$ divides the space into a disjoint union of regions $P_1, P_2, \ldots$, such that each P_i, called a regular region, has a volume of b and a diameter $\leq c$. A (b,c)-regular point set A is a set of points in $\Re^d$ with a (b,c)-partition $P_1, P_2, \ldots$, such that each P_i contains at most one point from A. For two regions A and B, if $A \subseteq B$ ($A \cap B \neq \emptyset$), we say B contains (intersects resp.) A.*

For the case $b = 1$ and $c = \sqrt{2}$, the plane can be partitioned into 1×1 squares, where each 1×1-square is a region $\{(x,y) | i \leq x < i+1 \text{ and } j \leq y < j+1\}$ for some grid point (i,j) with two integers i and j. All grid points are $(1, \sqrt{2})$-regular points.

Let $B_d(r,o)$ be the d-dimensional ball of radius r at center o. Its volume is $V_d(r) = \frac{2^{(d+1)/2}\pi^{(d-1)/2}}{1 \cdot 3 \cdots (d-2) \cdot d} r^d$ if d is odd, or $\frac{2^{d/2}\pi^{d/2}}{2 \cdot 4 \cdots (d-2) \cdot d} r^d$ otherwise.Let $V_d(r) = v_d \cdot r^d$, where v_d is a constant for the fixed dimension d. In particular, $v_1 = 2, v_2 = \pi$ and $v_3 = \frac{4\pi}{3}$. We will use the following well-known fact that can be easily derived from Helly Theorem (see [19]).

Lemma 1. *For an n-element set P in the d-dimensional space $\Re^d$, there is a point q with the property that any half-space that does not contain q, covers at most $\frac{d}{d+1}n$ elements of P. (Such a point q is called a centerpoint of P.)*

Definition 3. *Let $a > 0$, p and o be two points in $\Re^d$. Define $Pr_d(a, p_0, p)$ to be the probability that the point p has $\leq a$ perpendicular distance to a random hyperplane L through the point p_0. Define function $f_{a,p,o}(L) = 1$ if p has a distance $\leq a$ to the hyperplane L through o, or 0 otherwise. The expectation of function $f_{a,p,o}(L)$ is $E(f_{a,p,o}(L)) = Pr_d(a, o, p)$. Assume $P = \{p_1, p_2, \ldots, p_n\}$ is a set of n points in $\Re^d$ and each p_i has weight $w(p_i) \geq 0$. Define function $F_{a,P,o}(L) = \sum_{p \in P} w(p) f_{a,p,o}(L)$ and function $U(P, L, a) = \sum_{p \in P} f_{a,p,o}(L)$.*

We give an upper bound for the expectation $E(F_{a,P,o}(L))$ for $F_{a,P,o}(L)$ in the lemma below.

Lemma 2 ([16]). *Let $d \geq 2$. Let o be a point in $\Re^d$, $a, b, c > 0$ be constants and $\epsilon, \delta > 0$ be small constants. Assume that $P_1, P_2, \ldots,$ form a (b,c)-partition for $\Re^d$, and the weights $w_1 > \cdots > w_k > 0$ satisfy $k \cdot w_1 = O(n^\epsilon)$. Let P be a set of n weighted (b,c)-regular points in a d-dimensional plane with $w(p) \in \{w_1, \ldots, w_k\}$ for each $p \in P$. Let n_j be the number of points $p \in P$ with $w(p) = w_j$ for $j = 1, \ldots, k$. Then we have $E(F_{a,P,o}(L)) \leq (k_d \cdot (\frac{1}{b})^{\frac{1}{d}} + \delta) \cdot a \cdot \sum_{j=1}^{k} w_j \cdot n_j^{\frac{d-1}{d}} + O(n^{\frac{d-2}{d}+\epsilon})$, where $k_d = \frac{2v_{d-1}}{v_d}$, v_d is constant for fixed dimensional number d. In particular, $k_2 = \frac{4}{\sqrt{\pi}}$ and $k_3 = \frac{3}{2}\left(\frac{4\pi}{3}\right)^{\frac{1}{3}}$.*

Definition 4. *For each point q and a hyperplane L in $\Re^d$, define $sd(q, L)$ to be the signed distance from q to L, which is $sd(q, L) = (q - q_0) \cdot v_L$, where "$\cdot$" is the regular inner product, q_0 is a point on L, and v_L is the normal vector of the hyperplane L with the first nonzero coordinate to be positive.*

Definition 5. *For a hyperplane L in $\Re^d$, if L is through a point q_0 and has the normal vector v, then it has linear equation $(u - q_0) \cdot v = 0$. If $q \in \Re^d$ and $d = sd(q, L)$, then the hyperplane L' through q and parallel to L has equation $(u - (q_0 + dv)) \cdot v = 0$. We use $L(d)$ to represent such a hyperplane L'.*

For an interval $I \subseteq R$, $\|I\|$ is the length of I. For example, $\|[a, b)\| = b - a$. We often use $Pr(E)$ to represent the probability of an event E. For a real number x, $\lfloor x \rfloor$ is the largest integer $y \leq x$, and $\lceil x \rceil$ are the least integer $z \geq x$. For an interval $[a, b] \subseteq R$, define $center([a, b])$ to be $\frac{a+b}{2}$.

3 The Improved Sublinear Time Randomized Algorithm

Theorem 1 ([21]). *Let $X_1, \cdots, X_n$ be n independent random 0-1 variables, where X_i takes 1 with probability at most p. Let $X = \sum_{i=1}^{n} X_i$. Then for any $\delta > 0$, $\Pr(X > (1 + \delta)pn) < \left[\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right]^{pn}$.*

Theorem 2 ([21]). *Let $X_1, \cdots, X_n$ be n independent random 0-1 variables, where X_i takes 1 with probability at least p. Let $X = \sum_{i=1}^{n} X_i$. Then for any $\delta > 0$, $\Pr(X < (1 - \delta)pn) < e^{-\frac{1}{2}\delta^2 pn}$.*

Define $\tau_1(\delta) = e^{-\frac{1}{2}\delta^2}$ and $\tau_2(\delta) = \frac{e^\delta}{(1+\delta)^{(1+\delta)}}$. Define $\tau(\delta) = \max(\tau_1(\delta), \tau_2(\delta))$. We note that $\tau_2(\delta)$ is always strictly less than 1 for all $\delta > 0$, and $\tau_2(\delta)$ is fixed if δ is a constant. This can be verified by checking that the function $f(x) = (1 + x)\ln(1 + x) - x$ is increasing and $f(0) = 0$. This is because $f'(x) = \ln(1 + x)$ which is strictly greater than 0 for all $x > 0$.

Theorem 3. *Let $d \geq 2$ be the fixed dimension number and v be a positive parameter. Let $a, b, c > 0$ be constants and $\delta, s_1, s_2 > 0$ be small constants. Let Q be a set of n_Q points in $\Re^d$, and P be a set of n_P (b, c)-regular points, which form a sketch for Q. Then there exists an $O(n^{\frac{1}{d}} \cdot (\log n)^{1+\epsilon})$ time randomized algorithm to find a hyperplane L with probability $\geq 1 - \frac{1}{2^v}$ such that (1) each half space has $\leq (\frac{d}{d+1} + \delta)n_Q$ points from Q, and (2) and the number of points of P with distance $\leq a$ to L is $\leq (1 + \delta)k_d b^{\frac{-1}{d}} \cdot a \cdot n_P^{\frac{d-1}{d}}$, where $n = n_P + n_Q$ and ϵ is an arbitrary small constant.*

Proof (for Theorem 3). We use two phases to find the separator hyperplane. The first phase determines the orientation of the hyperplane by selecting a random hyperplane, and finds the region of the separator hyperplane for a balanced partition. The second phase finds the position of the separator plane with a small number of points in P close to it. Without loss of generality, we assume that $0 < \delta < 1$. Select constant $c_0 > 0$ and let $\delta_1 = c_0\delta$ so that

$$(k_d \cdot b^{\frac{-1}{d}} + 3\delta_1)(1 + \delta_1)^2 \leq (k_d \cdot b^{\frac{-1}{d}} + \frac{\delta}{2}). \tag{1}$$

Let $a_1 = a(1 + \delta_1)$ and $\alpha = \delta_1$.

Let o be the center point from Lemma 1 (our algorithm does not need to find such a center point o, but will use its existence). We consider all points in P having the same weight 1. By Lemma 2, we have

$$E(F_{a_1,P,o}) \leq (k_d \cdot b^{\frac{-1}{d}} + \delta_1) \cdot a_1 \cdot n_P^{\frac{d-1}{d}} + O(n_P^{\frac{d-2}{d}+\epsilon}). \tag{2}$$

By the well known Markov inequality and inequality (2),

$$Pr(F_{a_1,P,o}(L) \geq (1+\alpha)E(F_{a_1,P,o})) \leq \frac{1}{1+\alpha}. \tag{3}$$

This tells us that for a random hyperplane L, the probability is at least $1 - \frac{1}{1+\alpha}$ such that there exists a separator hyperplane L' (it may be through o) that satisfies the conditions of the theorem and is parallel to L. The hyperplane L' is determined by the signed distance from a point in L' to the hyperplane L since L' and L are parallel. We assign the values to some parameters:

$$r = c_4 v, \text{where } c_4 \text{ is a constant to be fixed later} \tag{4}$$

$$\epsilon_0 = \frac{\delta}{7} \tag{5}$$

$$\epsilon_1 = 5\epsilon_0 \tag{6}$$

$$m_1 = \frac{3(\ln 100 + r + \log n_Q)}{\epsilon_0^2} \tag{7}$$

$$\epsilon_c > 0 \text{ is a very small constant} \tag{8}$$

$$\epsilon \quad \text{is an arbitrary constant} > 0 \tag{9}$$

$$\xi \quad \text{is a positive constant with} \tag{10}$$

$$\frac{1+\xi}{1-\xi}(1+\alpha)(k_d \cdot b^{\frac{-1}{d}} + \delta_1) \leq (1+\delta)k_d \cdot b^{\frac{-1}{d}} \tag{11}$$

$$m_2 = 8c_2 r n_P^{\frac{1}{d}} (\log n_P)^{1+\epsilon}, \text{ where } c_2 \text{ is a constant with } \tau(\xi)^{c_2} \leq e \tag{12}$$

Algorithm: find separator in d-dimension

Input:

P (a set of weighted (b,c)-regular points in $\Re^d$),
Q (a set of points in $\Re^d$),
$n_P = |P|$ (the number of elements of set P), and
$n_Q = |Q|$ (the number of elements of set Q).

Phase 1:

begin

Select a fixed point $o^* \in \Re^d$ and a random hyperplane L through o^*.
Randomly select a list m_1 points $Q' =< q_1, \cdots, q_{m_1} >$ from Q.
For each $q_j \in Q'$, compute its signed distance to L $d_{q_i} = sd(q_i, L)$.
Find the $\lfloor(\frac{1}{d+1} - \epsilon_1)m_1\rfloor$-th least point $D^*_{1,d+1} = d_{q_1^*}$ among $d_{q_1}, \cdots, d_{q_{m_1}}$.
Find the $\lceil(\frac{d}{d+1} + \epsilon_1)m_1\rceil$-th least point $D^*_{d,d+1} = d_{q_2^*}$ among $d_{q_1}, \cdots, d_{q_{m_1}}$.
Randomly select a list of m_2 points $P' =< p_1, \cdots, p_{m_2} >$ from P.
For each $p_i \in P'$, compute $d_{p_i} = sd(p_i, L)$.

end (Phase 1)

Phase 2:
begin
if ($|D^*_{1,d+1} - D^*_{d,d+1}| \geq 3an_P^{\frac{1}{d}}(\log n_P)^{\epsilon}$) then (Case 1)
begin
Let $u = n_P^{\frac{1}{d}}(\log n_P)^{\epsilon}$.
Partition $[D^*_{1,d+1}, D^*_{d,d+1}]$ into equal length intervals $[l_1, l_2), [l_2, l_3), \cdots, [l_{u-1}, l_u), [l_u, l_{u+1}]$.
Compute $W(P', L, [l_i, l_{i+1}])$ for $i = 1, \cdots, u$.
Select $[l_i, l_{i+1}]$ with the minimal $W(P', L, [l_i, l_{i+1}])$.
end (Case 1)
if ($|D^*_{1,d+1} - D^*_{d,d+1}| \leq \delta_1 a$) then (Case 2: Subcase 2.1)
begin
Select $J = [D^*_{1,d+1} - a, D^*_{1,d+1} + a]$.
end (Case 2: Subcase 2.1)
if ($\delta_1 a < |D^*_{1,d+1} - D^*_{d,d+1}| < 3an_P^{\frac{1}{d}}(\log n_P)^{\epsilon}$) then (Case 2: Subcase 2.2)
begin
Select the least integer $v \geq 2$ such that $\frac{|D^*_{d,d+1} - D^*_{1,d+1}| + 2a}{v} \leq \frac{\delta_1 a}{3}$.
Let $s = \frac{|D^*_{d,d+1} - D^*_{1,d+1}| + 2a}{v}$.
Partition $[D^*_{1,d+1} - a, D^*_{d,d+1} + a]$ into $[r_1, r_2) \cup [r_2, r_3) \cup \cdots \cup [r_{v-1}, r_v) \cup [r_v, r_{v+1}]$ of length s.
Compute $W(P', L, I_i)$ with $I_i = [r_i, r_{i+1})$ for $i = 1, \cdots, v-1$ and $I_v = [r_v, r_{v+1}]$.
Select an integer h with $2a < h \cdot s < 2a + 2s$.
Let $J_i^* = [r_i, r_{i+h}) = I_i \cup I_{i+1} \cup \cdots \cup I_{i+h-1}$ $(i = 1, 2, \cdots, v-h)$ and
$J^*_{v-h+1} = [r_{v-h+1}, r_{v+1}] = I_{v-h+1} \cup I_{v-h+2} \cup \cdots \cup I_{v+1}$.
Compute $W(P', L, J_i^*)$ via $W(P', L, J_i^*) = W(P', L, J^*_{i-1}) - W(P', L, I_{i-1}) + W(P', L, I_{i+h})$ $(i = 1, \cdots, v-h+1)$.
Select $J = J_i^*$ with the minimal $W(P', L, J_i^*)$.
end (Case 2: Subcase 2.2)
Output $L(center(J))$ (see definition 5) as the separator hyperplane.
end (Phase 2)
Repeat the algorithm z times to amplify probability.
End of the Algorithm

Phase 1 of the algorithm: The input of our algorithm is $P, Q, n_Q = |Q|$, and $n_P = |P|$. Each input point $p \in P$ has the format $(x_1, \cdots, x_d)$. The algorithm starts with the following steps: Select a fixed point $o^* \in \Re^d$ and a random plane L through o^* (random hyperplane can be selected via selecting a random normal vector). Randomly select m_1 points $q_1, \cdots, q_{m_1}$ from Q and let $Q' = < q_1, \cdots, q_{m_1} >$ represent the list of these points just selected from Q (One point may appear multiple times. This is why we use list instead of set). For each $q_j \in Q'$, compute its signed distance $d_{q_i} = sd(q_i, L)$ to L. Find the $\lfloor(\frac{1}{d+1} - \epsilon_1)m_1\rfloor$-th least point $D^*_{1,d+1} = sd(q_1^*, L)$ for $d_{q_1}, \cdots, d_{q_{m_1}}$. Find the $\lceil(\frac{d}{d+1} + \epsilon_1)m_1\rceil$-th least point

$D^*_{d,d+1} = sd(q^*_2, L)$ for $d_{q_1}, \cdots, d_{q_{m_1}}$. Randomly select m_2 points $p_1, \cdots, p_{m_2}$ from P and let $P' =< p_1, \cdots, p_{m_2} >$ represent the list of these points just selected. For each $p_i \in P'$, compute $d_{p_i} = sd(p_i, L)$. It is well-known that finding the i-th element from a list takes linear steps (see [25]). The computation above takes $O(m_1 + m_2)$ steps. In the rest of the algorithm, we locate the position of the separator hyperplane parallel to L by finding its signed distance to L. Its position will be at the center of an interval of size $2a$. In the rest of the proof, we treat both P and Q as lists of points from $\Re^d$. Each point appears only at most once on both P and Q. Let $t_d = k_d \cdot b^{\frac{-1}{d}} + \delta$. For $q \in \Re^d$ and $A \subseteq \Re^d$, define $Pr(A, L, \leftarrow q) = \frac{|\{q' | q' \in A \text{ and } sd(q', L) \leq sd(q, L)\}|}{|A|}$. For a list of points $B =< x_1, \cdots, x_m >$ from $\Re^d$ and a point $q \in \Re^d$, define $X_{B,L,q}(i) = 1$ if $sd(x_i, L) \leq sd(q, L)$, or 0 otherwise. We also define $Y(B, L, q) = \sum_{i=1}^{m} X_{B,L,q}(i)$.

Lemma 3 ([1]). *With failure probability at most* $\frac{e^{-r}}{50}$*, we have that* $Pr(Q, L, \leftarrow q^*_1) \in [\frac{1}{d+1} - \delta, \frac{1}{d+1} - \frac{\delta}{6}]$ *and* $Pr(Q, L, \leftarrow q^*_2) \in [\frac{d}{d+1} + \frac{\delta}{6}, \frac{d}{d+1} + \delta]$ *for all large* n_Q.

Phase 2 of the algorithm: In this phase, we will find a position of the hyperplane L' (parallel to the hyperplane L) with the signed distance to L in the range $[D^*_{1,d+1}, D^*_{d,d+1}]$. Lemma 3 guarantees (with high probability) that each position in the interval $[D^*_{1,d+1}, D^*_{d,d+1}]$ gives a balance partition. We look for the position that has the small number of points in P close to L'.

For a list $A =< x_1, \cdots, x_m >$, $|A| = m$ is denoted to be the *length of* A and $x \in A$ means that x is one of the elements in A ($x = x_i$ for some $1 \leq i \leq m$). For a real number subset $J \subseteq \Re$ and a list A of finite points in $\Re^d$, define

$$Pr_*(A, L, J) = \frac{|\{p : p \in A \text{ and } sd(p, L) \in J\}|}{|A|}, \tag{13}$$

and also define

$$W(A, L, J) = |\{p : p \in A \text{ and } sd(p, L) \in J\}|. \tag{14}$$

In the proofs of the next a few lemmas, we pay much attention to handle the intervals H_i with $W(P, L, H_i) = \Omega(n_P^{\frac{d-1}{d}})$ and ignore those H_i with $W(P, L, H_i) = o(n_P^{\frac{d-1}{d}})$. This is because any interval H_i with $W(P, L, H_i) = o(n_P^{\frac{d-1}{d}})$ can directly bring an position of separator L with the number of points with distance to L at most a to be bounded by $(1 + \delta)k_d b^{\frac{-1}{d}} \cdot a \cdot n^{\frac{d-1}{d}}$.

Lemma 4. *Assume that* H_i *is an interval with* $W(P, L, H_i) = o(n_P^{\frac{d-1}{d}})$ *and* H_j *is an interval with* $W(P, L, H_j) = \Omega(n_P^{\frac{d-1}{d}})$*. Then with probability at most* $\frac{1}{100 n_P} e^{-r}$, $W(P', L, H_i) \geq W(P', L, H_j)$.

Proof. It follows from Theorems 1, 2 and the setting for m_2 by equation (12).

Lemma 5. *Let $f \leq n_P$ be an integer and $H_1, H_2, \cdots, H_f \subseteq R$ be f real intervals. With failure probability at most $\frac{1}{100}e^{-r}$, we have that $W(P,L,H_i) \in [W(P',L,H_i)\frac{n_P}{(1+\xi)m_2}, W(P',L,H_i)\frac{n_P}{(1-\xi)m_2}]$ for each interval H_i with $W(P,L,H_i) = \Omega(n_P^{\frac{d-1}{d}})$ and $i \leq f$.*

Proof. Assume that H_i is an fixed interval with $W(P,L,H_i) = \Omega(n^{\frac{d-1}{d}})$. We have $Pr_*(P,L,H_i) = \Omega(n_P^{\frac{-1}{d}})$ by the equation (13). By Theorems 1 and 2, the probability is at most $2\tau(\xi)^{-Pr_*(P,L,H_i)m_2} \leq \frac{e^{-r}}{100n_P}$ (see equation (12)) that we do not have that $W(P',L,H_i) \in [(1-\xi)Pr_*(P,L,H_i)m_2, (1+\xi)Pr_*(P,L,H_i)m_2]$. Thus, it has the probability $\leq f \cdot \frac{1}{100n_P}e^{-r} \leq \frac{1}{100}e^{-r}$, we do not have that $W(P',L,H_i) \in [(1-\xi)Pr_*(P,L,H_i)m_2, (1+\xi)Pr_*(P,L,H_i)m_2]$ for each $i \leq f$ with $W(P,L,H_i) = \Omega(n^{\frac{d-1}{d}})$. We assume that for all H_i with $i \leq f$ and $W(P,L,H_i) = \Omega(n^{\frac{d-1}{d}})$,

$$W(P',L,H_i) \in [(1-\xi)Pr_*(P,L,H_i)m_2, (1+\xi)Pr_*(P,L,H_i)m_2]. \quad (15)$$

Since $W(P,L,H_i) = Pr_*(P,L,H_i)n_P$ (by equations (13) and (14)) and assumption (15), we have $W(P,L,H_i) \in [W(P',L,H_i)\frac{n_P}{(1+\xi)m_2}$, $W(P',L,H_i)\frac{n_P}{(1-\xi)m_2}]$. We have proved the lemma. □

Case 1: $|D^*_{1,d+1} - D^*_{d,d+1}| \geq 3an_P^{\frac{1}{d}}(\log n_P)^\epsilon$. Partition $[D^*_{1,d+1}, D^*_{d,d+1}]$ into disjoint intervals $[l_1,l_2), [l_2,l_3), \cdots, [l_{u-1},l_u), [l_u,l_{u+1}]$ such that each $l_{i+1} - l_i (i = 1, \cdots, u)$ is equal to $\frac{|D^*_{1,d+1} - D^*_{d,d+1}|}{g_1(n_P)} \geq 3a$, where $g_1(n_P) = u = n_P^{\frac{1}{d}}(\log n_P)^\epsilon$. Let $J_i = [l_i, l_{i+1})$ if $i < u$, and $J_u = [l_u, l_{u+1}]$. Compute $W(P',L,J_i)$ for $i = 1, \cdots, u$, which takes $O(m_2 + g_1(n_P)) = O(m_2)$ steps. The algorithm selects $J = J_{i_0}$ that has the least $W(P',L,J_{i_0})$ and let $L' = L(center(J_{i_0}))$ (see definition 5), which takes $O(g_1(n_P)) = O(m_2)$ steps. Assume that J_{i_1} is the interval with the least $W(P,L,J_{i_1})$.

Lemma 6. *Assume Case 1 condition is true. With failure probability at most $\frac{1}{50}e^{-r}$, we have that $W(P,L,J_{i_0}) \leq \left(\frac{1+\xi}{1-\xi}k_d \cdot b^{\frac{-1}{d}}\right) \cdot a \cdot n_P^{\frac{d-1}{d}}$ for all large n_P.*

Proof. Since $\sum_{j=1}^{k} n_j = n_P$, and $J_1, J_2, \cdots, J_u$ are disjoint intervals, we have $\sum_{i=1}^{g_1(n_P)} W(P,L,J_i) \leq W(P,L,(-\infty,+\infty)) = n_P$. Since J_{i_1} is the interval with the least $W(P,L,J_{i_1})$, we have

$$W(P,L,J_{i_1}) \leq \frac{n_P}{g_1(n_P)} \leq \left(k_d \cdot b^{\frac{-1}{d}}\right) \cdot a \cdot n_P^{\frac{d-1}{d}}. \quad (16)$$

As we mentioned before, we do not discuss the case that there is an interval J_i with $W(P,L,J_i) = o(n^{\frac{d-1}{d}})$. In this case, it's unlikely that some J_k with $W(P,L,J_k) = \Omega(n^{\frac{d-1}{d}})$ will have $W(P',L,J_k) \leq W(P',L,J_i)$ by Lemma 4.

By Lemma 5, with probability $\leq \frac{1}{100}e^{-r}$, we do not have that for each J_i with $W(P,L,J_i) = \Omega(n^{\frac{d-1}{d}})$ and $i \leq g_1(n_P)$,

$$W(P,L,J_i) \in [\ W(P',L,J_i)\frac{n_P}{(1+\xi)m_2}, W(P',L,J_i)\frac{n_P}{(1-\xi)m_2}]. \quad (17)$$

Assume (17) is true. Thus, $W(P', L, J_{i_1})\frac{n_P}{(1+\xi)m_2} \leq W(P, L, J_{i_1})$, which implies the following:

$$W(P', L, J_{i_1}) \leq W(P, L, J_{i_1})\frac{(1+\xi)m_2}{n_P}. \tag{18}$$

Since the algorithm selects the interval J_{i_0} with the least $W(P', L, J_{i_0})$, we have that

$$W(P', L, J_{i_0}) \leq W(P', L, J_{i_1}). \tag{19}$$

Thus, we conclude that $W(P, L, J_{i_0}) \leq W(P', L, J_{i_0})\frac{n_P}{(1-\xi)m_2} \leq W(P', L, J_{i_1})\frac{n_P}{(1-\xi)m_2} \leq ((1+\xi)W(P, L, J_{i_1})\frac{m_2}{n_P})\frac{n_P}{(1-\xi)m_2} = \frac{1+\xi}{1-\xi}W(P, L, J_{i_1})$.

By (16), we have $W(P, L, J_{i_0}) \leq \left(\frac{1+\xi}{1-\xi}k_d \cdot b^{\frac{-1}{d}}\right) \cdot a \cdot n_P^{\frac{d-1}{d}}$. □

Case 2: $|D^*_{1,d+1} - D^*_{d,d+1}| < 3an_P^{\frac{1}{d}}(\log n_P)^{\epsilon}$. Let J^* be interval such that $center(J^*) \in [D^*_{1,d+1}, D^*_{d,d+1}]$ and $|J^*| = 2a_1 = 2a(1+\delta_1)$ and $W(P, L, J^*)$ is the least.

Subcase 2.1: $|D^*_{1,d+1} - D^*_{d,d+1}| \leq \delta_1 a$. Let $J = [D^*_{1,d+1} - a, D^*_{1,d+1} + a]$ and let $L' = L(D^*_{1,d+1})$ (In other words, $L' = L(center(J))$) (see definition 5). Clearly, $J \subseteq J^*$ and $W(P, L, J) \leq W(P, L, J^*)$.

Subcase 2.2: $\delta_1 a < |D^*_{1,d+1} - D^*_{d,d+1}| < 3an_P^{\frac{1}{d}}(\log n_P)^{\epsilon}$. Let $g_2(n_P)$ be the least integer $v \geq 2$ such that $\frac{|D^*_{d,d+1} - D^*_{1,d+1}|+2a}{v} \leq \frac{\delta_1 a}{3}$. Since $v \geq 2$ and $\frac{|D^*_{d,d+1} - D^*_{1,d+1}|+2a}{v-1} > \frac{\delta_1 a}{3}$, we have $\frac{|D^*_{d,d+1} - D^*_{1,d+1}|+2a}{v} = \frac{v-1}{v}\frac{|D^*_{d,d+1} - D^*_{1,d+1}|+2a}{v-1} > \frac{v-1}{v}\frac{\delta_1 a}{3} \geq \frac{\delta_1 a}{6}$. Therefore, $v \leq \frac{|D^*_{d,d+1} - D^*_{1,d+1}|+2a}{\frac{\delta_1 a}{6}} \leq \frac{3an_P^{\frac{1}{d}}(\log n_P)^{\epsilon}+2a}{\frac{\delta_1 a}{6}} = \frac{6(3n_P^{\frac{1}{d}}(\log n_P)^{\epsilon}+2)}{\delta_1} = O(n_P^{\frac{1}{d}}(\log n_P)^{\epsilon})$. Let $s = \frac{|D^*_{d,d+1} - D^*_{1,d+1}|+2a}{g_2(n_P)} \in [\frac{\delta_1 a}{6}, \frac{\delta_1 a}{3}]$. Partition $[D^*_{1,d+1} - a, D^*_{d,d+1} + a]$ into the union of $g_2(n_P)$ disjoint intervals of size s: $[r_1, r_2) \cup [r_2, r_3) \cup \cdots \cup [r_{v-1}, r_v) \cup [r_v, r_{v+1}]$, where $v = g_2(n_P)$ and $r_{i+1} = r_i + s$ for $i = 1, \cdots, v$. Let $I_i = [r_i, r_{i+1})$ for $i = 1, \cdots, v-1$ and $I_v = [r_v, r_{v+1}]$. Let $J^*_i = I_i \cup I_{i+1} \cdots \cup I_{i+h-1}$ for $i = 1, \ldots, v-h+1$, where h is an integer with $2a < h \cdot s < 2a + 2s$. The algorithm selects the interval $J = J^*_{i_2}$ that has the least $W(P', L, J^*_{i_2})$. Finally, the algorithm outputs $L' = L(center(J))$ (see definition 5) for the separator hyper-plane. We analyze the algorithm for the case 2.

Lemma 7. *Assume that J is the interval output from the case 2 (either subcase 2.1 or subcase 2.2). With failure probability at most $\frac{1}{100}e^{-r}$, we have that $W(P, L, J) \leq \frac{1+\xi}{1-\xi}W(P, L, J^*)$.*

Proof. The subcase 2.1 is trivial since the small size of the interval implies that $J \subseteq J^*$. We only discuss the subcase 2.2. Let $I_t, I_{t+1}, \cdots, I_{t+m}$ be the intervals such that $J^* \cap I_{t+i} \neq \emptyset$ $(i = 0, \cdots, m)$. Then $I_{t+1}, I_{t+2}, \cdots, I_{t+m-1}$ are all

subsets of J^*. Since $\|I_i\| = s \leq \frac{\delta_1 a}{3}$, $\|J^*\| = 2(1+\delta_1)a \geq \|J^*_{t+1}\| \geq \|J^*\| - \delta_1 a - \|I_t\| - \|I_{t+m}\| \geq 2(1+\delta_1)a - \delta_1 a - \frac{2\delta_1 a}{3} \geq 2a + \frac{\delta_1 a}{3}$ (Remember that we use $\|[a,b)\|$ to represent the length b-a of the interval $[a,b)$). We have the interval J^*_{t+1} with $\|J^*_{t+1}\| \geq 2a$ and $J^*_{t+1} \subseteq J^*$. This implies that

$$W(P, L, J^*_{t+1}) \leq W(P, L, J^*). \tag{20}$$

By Lemma 5, it has probability at most $\frac{1}{100}e^{-r}$, the following does not hold: $W(P, L, J^*_i) \in [(1-\xi)W(P', L, J^*_i)\frac{n_P}{m_2}, (1+\xi)W(P', L, J^*_i)\frac{n_P}{m_2}]$ for each $i \leq g_2(n_P) - h + 1$ with $W(P, L, J^*_i) = \Omega(n_P^{\frac{d-1}{d}})$.

We assume that $W(P, L, J^*_i) \in [W(P', L, J^*_i)\frac{n_P}{(1+\xi)m_2}, W(P', L, J^*_i)\frac{n_P}{(1-\xi)m_2}]$ for all $i \leq g(n_P) - h + 1$. Thus, $W(P', L, J^*_{t+1})\frac{n_P}{(1+\xi)m_2} \leq W(P, L, J^*_{t+1}) \leq W(P, L, J^*)$. Hence, $W(P', L, J^*_{t+1}) \leq W(P, L, J^*_{t+1})\frac{(1+\xi)m_2}{n_P}$.

Since the algorithm selects the interval $J^*_{i_2}$ with the least $W(P', L, J^*_{i_2})$, we have $W(P', L, J^*_{i_2}) \leq W(P', L, J^*_{t+1})$. We have that $W(P, L, J^*_{i_2}) \leq W(P', L, J^*_{i_2})\frac{n_P}{(1-\xi)m_2} \leq W(P', L, J^*_{t+1})\frac{n_P}{(1-\xi)m_2} \leq ((1+\xi)W(P, L, J^*_{t+1})\frac{m_2}{n_P})\frac{n_P}{(1-\xi)m_2} = \frac{1+\xi}{1-\xi}W(P, L, J^*_{t+1}) \leq \frac{1+\xi}{1-\xi}W(P, L, J^*)$.

As we mentioned before, we do not discuss the case that there is an interval J_i with $W(P, L, J_i) = o(n^{\frac{d-1}{d}})$. In this case, it's unlikely that some J_k with $W(P, L, J_k) = \Omega(n^{\frac{d-1}{d}})$ will have $W(P', L, J_k) \leq W(P', L, J_i)$ by Lemma 4. □

For a list A of finite points in $\Re^d$ and a hyper-plane M_1, define $F_1(M_1, a, A) = \sum_{p_i \in A \text{ and } \text{dist}(p_i, M_1) \leq a} w(p_i)$. If M_1 and M_2 are two parallel hyper-planes with signed distance $d_{M_1,M_2} = sd(p, M_1)$ for some point p in the M_2, then $F_1(M_2, a, A) = W(A, M_1, [d_{M_1,M_2}, -a, d_{M_1,M_2} + a])$. The the hyper-plane $L(center(J^*_{i_2}))$ (see definition 5) output by the algorithm has that $F_1(L(center(J)), a, P') \leq \frac{1+\xi}{1-\xi}F_1(L(center(J^*)), a_1, P')$ by Lemma 7.

Lemma 8. *With failure probability at most e^{-v}, one can output an hyperplane L' in $O(v^2 \cdot (n^{\frac{1}{d}} \cdot (\log n)^{1+\epsilon})$ steps such that $F_1(L', a, P) \leq \left((1+\delta)k_d \cdot b^{\frac{-1}{d}}\right) \cdot a \cdot n_P^{\frac{d-1}{d}}$, and each side of the half-space contains at most $(\frac{d}{d+1} + \delta)n_Q$ points in Q, where $n = n_P + n_Q$.*

Proof. After the hyper-plane L is selected in phase one, by Lemma 3 we have the probability at most e^{-r} that both $Pr(Q, L, \leftarrow q^*_1) \in [\frac{1}{d+1} - \delta, \frac{1}{d+1} - \frac{\delta}{6}]$ and $Pr(Q, L, \leftarrow q^*_2) \in [\frac{d}{d+1} + \frac{\delta}{6}, \frac{d}{d+1} + \delta]$. This means every L' (parallel to L) with the signed distance (to L) in the interval $[D^*_{1,d+1}, D^*_{d,d+1}]$, it has at most $(\frac{d}{d+1} + \delta)n_Q$ points of Q in each of the half spaces. In phase 2, we have probability at most e^{-r} that it does not output the separator L' (the signed distance to L is in $[D^*_{1,d+1}, D^*_{d,d+1}]$) such that $F_1(L', a, P) \leq \left(\frac{1+\xi}{1-\xi}k_d \cdot b^{\frac{-1}{d}}\right) \cdot a \cdot n_P^{\frac{d-1}{d}}$ (Case 1 of Phase 2, see Lemma 6) or $F_1(L', a, P)) \leq \frac{1+\xi}{1-\xi}F_1(L(J^*), a_1, P)$ (Case 2 of Phase 2, see Lemma 7), where J^* is the interval of length $2a_1$ with the least $F_1(L(J^*), a_1, P)$ and center between $D^*_{1,d+1}$ and $D^*_{d,d+1}$.

Assume that L is a fixed hyper-plane and L^* is a another hyper-plane that is parallel to L and $F_1(L^*, a_1, P)$ is the least. By Lemma 6 and Lemma 7, it has probability is at most e^{-r} such that we cannot get another L' (parallel to L) such that $F_1(L', a, P) \leq \left(\frac{1+\xi}{1-\xi} k_d \cdot b^{\frac{-1}{d}}\right) \cdot a \cdot n_P^{\frac{d-1}{d}}$ or $F_1(L', a, P) \leq \frac{1+\xi}{1-\xi} F_1(L^*, a_1, P)$.

With probability at most $\frac{1}{1+\alpha}$, $F_{a_1,P,o}(L) \geq (1+\alpha)E(F_{a_1,P,o})$ (by (3)). If the algorithm repeats z times, let $L_1, \cdots, L_z$ be the random hyper planes selected for L. With probability at most $(\frac{1}{1+\alpha})^z$, none of those L_is has another hyper-plane L_i^* such that L_i^* is parallel to L_i and has $F_{a_1,P,o}(L_i^*) \leq (1+\alpha)E(F_{a_1,P,o})$. Therefore, we have probability at most $(\frac{1}{\alpha+1})^z + 2ze^{-r}$ that we cannot find such a hyper-plane L' with

$$F_1(L', a, P) \leq \frac{1+\xi}{1-\xi}(1+\alpha)E(F_{a_1,P,o}) \text{ or} \tag{21}$$

$$F_1(L', a, P) \leq \left(\frac{1+\xi}{1-\xi} k_d \cdot b^{\frac{-1}{d}}\right) \cdot a \cdot n_P^{\frac{d-1}{d}}. \tag{22}$$

By inequalities (21), (22), (11) and (2), we have
$F_1(L', a, P) \leq \left(\frac{1+\xi}{1-\xi}(1+\alpha) k_d \cdot b^{\frac{-1}{d}}\right) \cdot a \cdot n_P^{\frac{d-1}{d}} \leq \left((1+\delta) k_d \cdot b^{\frac{-1}{d}}\right) \cdot a \cdot n_P^{\frac{d-1}{d}}$.

Now we give a bound for the probability. Let $z = \frac{2r}{\ln(1+\alpha)} = O(v)$ (by (4)). Then $(\frac{1}{1+\alpha})^z + 2ze^{-r} < 2^{-v}$, where we let $r = c_4 v$ for some constant c_4 large enough.

The phase 1 of the algorithm takes $O(m_1 + m_2)$ steps. The case 1 of phase 2 takes $O(m_2)$ steps. The case 2 of phase 2 takes $O(m_2)$ steps. Totally, it takes $O(z(m_1 + m_2)) = O(v^2 \cdot (n^{\frac{1}{d}} (\log n)^{1+\epsilon}))$ steps. □

Applying Lemma 8, we finish the proof of the Theorem. □

We have a more generalized version of algorithm for finding width bounded separator. It will be presented in the journal version of this paper. We also derive the following lower bound that almost matches the upper bound.

Theorem 4. *There is no $o(n^{\frac{1}{d}})$-time randomized algorithm that tests if a given hyper-plane S is a 2-width bounded separator for a set P of n grid points.*

References

1. Fu, B., Chen, Z.: Sublinear time width-bounded separators and their application to the protein side-chain packing problem. Journal of Combinatorial Optimization 15, 387–407 (2008)
2. Fu, B., Wang, W.: Geometric Separators and Their Applications to Protein Folding in the HP-Model. SIAM J. Comput. 37(4), 1014–1029 (2007)
3. Lipton, R.J., Tarjan, R.: A separator theorem for planar graph. SIAM J. Appl. Math. 36, 177–189 (1979)
4. Djidjev, H.N.: On the problem of partitioning planar graphs. SIAM Journal on Discrete Mathematics 3(2), 229–240 (1982)

5. Gazit, H.: An improved algorithm for separating a planar graph, USC (1986) (manuscript)
6. Alon, P.N., Thomas, R.: Planar separator. SIAM J. Discr. Math. 7(2), 184–193 (1990)
7. Djidjev, H.N., Venkatesan, S.M.: Reduced constants for simple cycle graph separation. Acta informatica 34, 231–234 (1997)
8. Spielman, D.A., Teng, S.H.: Disk packings and planar separators. In: The 12th annual ACM symposium on computational geometry, pp. 349–358 (1996)
9. Gilbert, J.R., Hutchinson, J.P., Tarjan, R.E.: A separation theorem for graphs of bounded genus. Journal of algorithm 5, 391–407 (1984)
10. Alon, N., Seymour, P., Thomas, R.: A separator theorem for graphs with an excluded minor and its applications. In: STOC, pp. 293–299 (1990)
11. Plotkin, S., Rao, S., Smith, W.D.: Shallow excluded minors and improved graph decomposition. In: SODA, pp. 462–470 (1990)
12. Miller, G.L., Teng, S.-H., Vavasis, S.A.: An unified geometric approach to graph separators. In: FOCS, pp. 538–547 (1991)
13. Miller, G.L., Thurston, W.: Separators in two and three dimensions. In: STOC, pp. 300–309 (1990)
14. Smith, W.D., Wormald, N.C.: Application of geometric separator theorems. In: FOCS, pp. 232–243 (1998)
15. Miller, G.L., Vavasis, S.A.: Density graphs and separators. In: SODA, pp. 331–336 (1991)
16. Fu, B.: Theory and application of width bounded geometric separator. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 277–288. Springer, Heidelberg (2006)
17. Chen, Z., Fu, B., Tang, Y., Zhu, B.: A PTAS for a disc covering problem using width-bounded geometric separator. J. of Comb. Opt. 11(2), 203–217 (2006)
18. Eppstein, D., Miller, G.L., Teng, S.-H.: A deterministic linear time algorithm for geometric separators and its applications. Fundam. Inform. 22(4), 309–329 (1995)
19. Pach, J., Agarwal, P.: Combinatorial Geometry. Wiley-Interscience Publication, Hoboken (1995)
20. Li, M., Ma, B., Wang, L.: On the closest string and substring problems. Journal of the ACM 49(2), 157–171 (2002)
21. Motwani, R., Raghavan, P.: Randomized Algorithms, Cambridge (2000)
22. Akutsu, T.: NP-hardness results for protein side-chain packing. In: Miyano, S., Takagi, T. (eds.) Genome Informatics, vol. 9, pp. 180–186 (1997)
23. Canutescu, A.A., Shelenkov, A.A., Jr., R.L.D.: A graph-theory algorithm for rapid protein side-chain prediction. Protein science 12, 2001–2014 (2003)
24. Chazelle, B., Kingsford, C., Singh, M.: A semidefinite programming approach to side-chain positioning with new rounding strategies. INFORMS Journal on Computing 16, 86–94 (2004)
25. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press, Cambridge (2001)
26. Ponter, J.W., Richards, F.M.: Tertiary templates for proteins: use of packing criteria and the enumeration of allowed sequences for different structural classes. Journal of molecular biology 193, 775–791 (1987)

Constant Time Generation of Biconnected Rooted Plane Graphs

Bingbing Zhuang and Hiroshi Nagamochi

Graduate School of Informatics, Kyoto University
{zbb,nag}@amp.i.kyoto-u.ac.jp

Abstract. A plane graph is a drawing of a planar graph in the plane such that no two edges cross each other. In a rooted plane graph, an outer (directed) edge is designated as the root. For a given positive integer $n \geq 1$, we give an $O(1)$-time delay algorithm that enumerates all plane graphs with exactly n vertices using $O(n)$ space. Our algorithm can generates only plane graphs such that the size of each inner face is bounded from above by a prescribed integer $g \geq 3$ in the same time and space complexity.

1 Introduction

The problem of enumerating (i.e., listing) all graphs with bounded size is one of the most fundamental and important issues in graph theory. Many algorithms for particular classes of graphs have been studied [1,11,13,14,18,20], and several excellent books on this subject have been published [3,9,19]. Cataloguing graphs, i.e., making the complete list of graphs in a particular class can be used in a various way: search for a possible counterexample to a mathematical conjecture; choosing the best graph among all candidate graphs; and experiment for measuring the average performance of a graph algorithm over all possible input graphs.

Time delay of an enumeration algorithm is a time bound between two consecutive outputs. Enumerating graphs with a polynomial time delay would be rather easy since we can examine the whole structure of the current graph anytime. However, algorithms with a constant time delay in the worst case is a hard target to achieve without a full understanding of the graphs to be enumerated, since not only the difference between two consecutive outputs is required to be $O(1)$, but also any operation for examining symmetry and identifying the edges/vertices to be modified to get the next output needs to be executable in $O(1)$ time. One of the common ideas behind efficient enumeration algorithms (e.g., [16,15,17]) is to define a unique representation for each graph in a graph class as its "parent," which induces a rooted tree that connects all graphs in the class, called the *family tree* $\mathcal{F}$, where each node in $\mathcal{F}$ corresponds to a graph in the class. Then all graphs in the class will be enumerated one by one according to the depth-first traversal of the family tree $\mathcal{F}$. However, the crucial point to attain an $O(1)$-time delay is to find a "good" parent which enables us to generate each of the children from a graph in $O(1)$ time.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 113–123, 2010.

Enumeration of restricted graphs or graphs with configurations has many applications in various fields such as machine learning and chemoinformatics. Enumeration of trees and outerplanar graphs can be used for many purposes including the inference of structures of chemical compounds [8], virtual exploration of chemical universe [12], and reconstruction of molecular structures from their signatures [4]. It is known that 94.3% of chemical compounds in NCI chemical database have planar structures [6]. Hence planar graphs is an important class to be investigated.

Our research group has been developing algorithms for enumerating chemical graphs that satisfy given various constraints [2,7,8]. We have designed efficient branch-and-bound algorithms for enumerating tree-like chemical graphs [2,8], which are based on the tree enumeration algorithm [16], and implementations of these algorithms are available on our web server[1]. Currently we aim to provide efficient algorithms for enumerating chemical graphs for a wider class of graphs than trees such as cacti and outerplanar graphs in our web server.

Li and Nakano [10] presented an efficient algorithm that enumerates all biconnected rooted triangulated plane graphs in constant time per each, where an outer edge is chosen as the root of each biconnected rooted plane graph. Afterwards Nakano [15] presented an algorithm with the same time complexity to generate all triconnected rooted triangulated plane graphs. Recently, in our companion paper [22], we gave an efficient algorithm for enumerating biconnected rooted planar graphs with internally triangulated faces, where a planar graph designates an outer vertex v and two outer edges incident to v.

Yamanaka and Nakano [21] gave an algorithm for generating all connected rooted plane graphs with at most m edges, where a plane graph has one designated (directed) edge on the outer face. The algorithm uses $O(m)$ space and generates such graphs in $O(1)$ time per graph on average without duplications.

In this paper, we consider the class $\mathcal{G}_2(n, g)$ of all biconnected rooted plane graphs with exactly n vertices such that the size of each inner face is at most g, where a rooted plane graph here designates an outer vertex. We give an algorithm that enumerates all plane graphs in $\mathcal{G}_2(n, g)$ in $O(1)$ time per graph in the worst case using $O(n)$ space. Our algorithm also yields an $O(n^3)$-time delay algorithm for generating all biconnected *unrooted* plane graphs with exactly n vertices such that the size of each inner face is at most g. However, our algorithm does not exploit any dynamic data structure that represents triconnected components to test biconnectivity of possible candidates for graphs to be generated, because an $O(1)$ time maintenance of such a data structure required to achieve an $O(1)$-time delay seems extremely difficult. To design an $O(1)$-time delay algorithm for $\mathcal{G}_2(n, g)$, we use structural properties of cut-pairs of outer vertices in biconnected plane graphs, and "balanced orientation" of all edges in a biconnected plane graph, which allows us to test whether given vertices u and v are adjacent or not in $O(1)$ time and $O(n)$ space. Note that such a test can be done in $O(1)$ time using an $O(n^2)$-space adjacency matrix and in time proportional to the sum of degrees of u and v using an $O(n)$-space adjacency lists.

[1] http://sunflower.kuicr.kyoto-u.ac.jp/tools/enumol/

The rest of the paper is organized as follows. After introducing basic notations in Section 2, Section 3 examines the structure of biconnected plane graphs. Section 4 introduces the parent of each biconnected rooted plane graph, and characterizes the children of a biconnected rooted plane graph. Sections 5 describes an algorithm for enumerating all biconnected rooted plane graphs, and analyzes the time and space complexities of the algorithm. Section 6 makes some concluding remarks. All proofs of the lemmas in this paper can be found in [23].

2 Preliminaries

Throughout the paper, a graph stands for a simple undirected graph unless stated otherwise. A graph is denoted by a pair $G = (V, E)$ of a vertex set V and an edge set E. The set of vertices and the set of edges of a given graph G are denoted by $V(G)$ and $E(G)$, respectively. For a subset $E' \subseteq E(G)$, $G - E'$ denotes the graph obtained from a graph G by removing the edges in E'. Let X be a subset of $V(G)$. We denote by $G - X$ the graph obtained from G by removing the vertices in X together with the edges incident with a vertex in X. Let $deg(v)$ denote the degree of a vertex v in a graph G.

A graph is called *planar* if its vertices and edges can be drawn as points and curves on the plane so that no two curves intersect except for their endpoints. A planar graph with such a fixed embedding is called a *plane* graph, where a face is designated as the *outer face* and all other faces are called *inner faces*. For a face f in a plane graph, let $V(f)$ and $E(f)$ denote the sets of vertices and edges on the facial cycle of f, and define the size $|f|$ of face f to be $|V(f)|$. A *rooted* plane graph is a plane graph which has a designated outer vertex r, which is called the *root*. Two rooted plane graphs G_1 and G_2 are *equivalent* if their vertex sets admit a bijection by which the root and the incidence-relation between edges and vertices/faces in G_1 correspond to those in G_2.

For given integers n and g, let $\mathcal{G}_2(n)$ denote the set of all biconnected rooted plane graphs with exactly n vertices, where no two plane graphs in $\mathcal{G}_2(n)$ are equivalent, and let $\mathcal{G}_2(n, g)$ denote the set of all graphs in $\mathcal{G}_2(n)$ such that the size of each inner face is at most g.

A plane graph G is called a *fan* if it is obtained from a path P with at least one vertex by adding a new vertex v together with an edge incident to each vertex in the path, where the vertex v is called the *center* of a fan. A fan with n vertices is denoted by F_n, and is treated as a plane graph rooted at the center.

For two outer vertices u and v in a biconnected plane graph, let $\beta[u, v]$ denote the path obtained by traversing the boundary of G from u to v in the clockwise order.

An *orientation* of an undirected graph G is to give an orientation for each undirected edge $\{u, v\}$ to obtain a digraph D_G, where $\{u, v\}$ becomes one of directed edge (u, v), directed edge (v, u) and a pair of directed edges (u, v) and (v, u). Let $E^+(v)$ denote the set of directed edges (v, u) in D_G. It should be noted that G contains an undirected edge $\{u, v\}$ if and only if $(u, v) \in E^+(u)$ or $(v, u) \in E^+(v)$ holds. An orientation is called *balanced* if $|E^+(v)| = O(|E(G)|/|V(G)|)$

holds for all vertices v in G. Note that a balanced orientation to a planar graph G satisfies $|E^+(v)| = O(1)$, $v \in V(G)$, implying that whether two given vertices u and v in G are adjacent or not can be tested in $O(|E^+(u)| + |E^+(v)|) = O(1)$ time and $O(|E(G)|) = O(n)$ space. In this paper, we prove that every planar graph admits a balanced orientation, which is one of the crucial points so that our algorithm can be implemented to run in $O(1)$-time delay in the worst case.

3 Biconnected Plane Graphs

In a biconnected plane graph G, an edge is called *2-removable* if $G - e$ remains biconnected, and is called *2-irremovable* otherwise. Let $e_1 = \{v_1, v_2\}$ be an outer edge in G, where v_2 appears immediately after v_1 along the boundary in the clockwise order, let f_1 be the inner face that contains edge e_1, and let $V^o(e_1)$ denote the set of all other outer vertices in f_1 than v_1 and v_2. Assume that e_1 is not 2-removable. Note that v_1 and v_2 are connected by the path Q from v_2 to v_1 along the boundary of G. Any cut-vertex x in $G - e_1$ must intersect path Q. Also v_1 and v_2 are connected by the path along inner facial cycle f_1. Hence the set of all cut-vertices in $G - e_1$ is given by $V^o(e_1) \neq \emptyset$. See Fig. 1(a). Let $\eta_{\mathtt{first}}(e_1)$ (resp., $\eta_{\mathtt{last}}(e_1)$) be the vertex in $V^o(e_1)$ that appears first (resp., last) when we traverse the boundary of G from v_2 in the clockwise order. Let $\beta(e_1)$ denote the path $\beta[v_2, \eta_{\mathtt{first}}(e_1)]$.

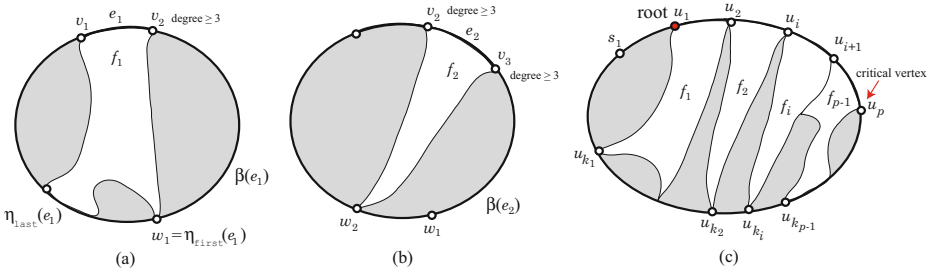


Fig. 1. (a) Vertices $\eta_{\mathtt{first}}(e_1)$ and $\eta_{\mathtt{last}}(e_1)$ for a 2-irremovable edge $e_1 = \{v_1, v_2\}$; (b) A 2-irremovable edge $e_2 = \{v_2, v_3\}$; (c) Edges e_i in the active path $\beta[r, u_p]$

Lemma 1. *Let $e_1 = \{v_1, v_2\}$ be a 2-irremovable outer edge in a biconnected plane graph G. Then $\beta(e_1)$ contains a 2-removable outer edge or an outer vertex of degree 2.*

By Lemma 1, the boundary of a biconnected plane graph G always contains a 2-removable outer edge or an outer vertex of degree 2. The *first removable element* is defined to be the first such edge or vertex that appears when we traverse the boundary of G from the root in the clockwise order.

Denote the sequence of vertices in the boundary of G by $u_1 = r, u_2, \ldots, u_B$, where u_{B+1} denotes $u_1 = r$, and denote the outer edge $\{u_i, u_{i+1}\}$ by e_i.

Consider the first removable element of G, which is an edge e or a vertex v, which we denoted by $\{u_p, u_{p+1}\}$ or u_p. We call the vertex u_p the *critical vertex*, and call the path $\beta[r, u_p]$ *active*. See Fig. 1(c). By Lemma 1, we see that each edge $e_i = \{u_i, u_{i+1}\}$ in the active path has $u_k = \eta_{\mathtt{last}}(e_i)$ such that $i+1 < k \leq B$. Let k_i denote the index k of such vertex $u_k = \eta_{\mathtt{last}}(e_i)$, where $k_1 = B$ if $deg(r) = 2$. let $k_0 = \infty$ for notational convenience. Clearly, it holds

$$k_0 \geq k_1 \geq k_2 \geq \cdots \geq k_{p-1} > p. \tag{1}$$

4 Parents of Biconnected Rooted Plane Graphs

A biconnected plane graph with $n \leq 3$ vertices is unique. In what follows, we assume that $n \geq 4$ and $g \geq 3$. Let G be a rooted plane graph, where the root r is an outer vertex. The neighbours of the root r of G are denoted by $s_1, s_2, \ldots, s_K$ ($K = deg(r)$) from the leftmost one to the rightmost one. The *fan factor* is defined to be the maximal subsequence $s_1, s_2, \ldots, s_t$ such that each s_i, $2 \leq i \leq t$ is adjacent to only s_{i-1}, s_{i+1} and r, and s_1 is adjacent to only s_2 and r. See Fig. 3(d), where $t = 3$. Let $\psi(G)$ denote the fan factor of G. If $\psi(G) \neq \emptyset$, then $\psi(G)$ induces F_t, $t = |\psi(G)|$ from G.

Let G be a biconnected rooted plane graph with $n \geq 4$ such that $G \neq F_n$. We define the *parent* $\mathcal{P}(G)$ of G to be the following graph with n vertices.

(i) If the first removable element is a vertex w adjacent to two neighbours u and v which are *not adjacent* each other, then $\mathcal{P}(G)$ is defined to be the graph obtained from G by replacing two edges $\{u, w\}$ and $\{w, v\}$ with a single edge $\{u, v\}$ and adding a new vertex v' as the leftmost neighbour of r together with two new edges $\{v', r\}$ and $\{v', s_1\}$. See Fig. 2(a) and (b).

(ii) If the first removable element is a vertex w adjacent to two neighbours u and v which are *adjacent* each other, then $\mathcal{P}(G)$ is defined to be the graph obtained from G by removing w and two edges $\{u, w\}$ and $\{w, v\}$ and adding a new vertex v' as the leftmost neighbour of r together with two new edges $\{v', r\}$ and $\{v', s_1\}$. Note that the edge joining u and v is not necessarily an outer edge. See Fig. 2(c) and (d).

(iii) If the first removable element is an edge e, then $\mathcal{P}(G)$ is defined to be $G - e$ (see Fig. 2(e) and (f)).

Let $\mathcal{P}^0(G) = G$ and $\mathcal{P}^i(G) = \mathcal{P}(\mathcal{P}^{i-1}(G))$ for integers $i \geq 1$.

Lemma 2. *For any graph $G \in \mathcal{G}_2(n)$ with $n \geq 3$, there is an integer $i \in [0, 3(n-2)]$ such that $\mathcal{P}^i(G) = F_n$.*

Let G be a biconnected rooted plane graph with $n \geq 4$ vertices. A rooted plane graph G' is called a *child* of G if $G = \mathcal{P}(G')$. Let $\mathcal{C}(G)$ denote the set of all children of G. Lemma 2 implies that the parent-child relationship by $\mathcal{P}$ forms a family tree rooted at node F_n. In order to generate all children of G, we introduce the following three operations.

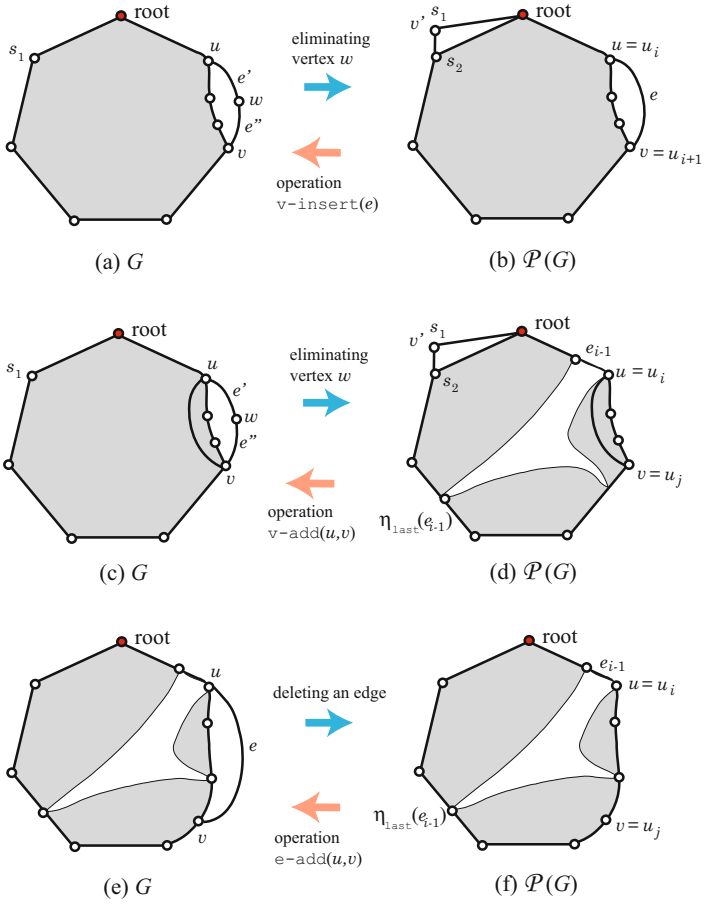
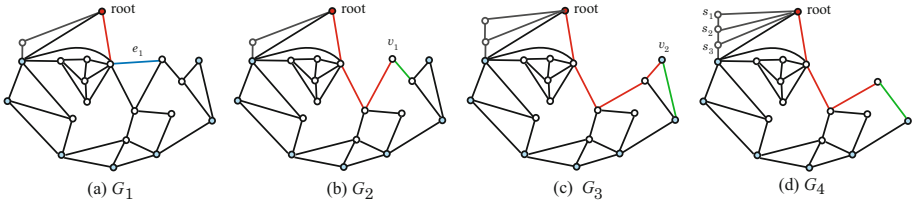


Fig. 2. (a),(c) and (e) A biconnected rooted plane graph G; (b), (d) and (f) The parent $\mathcal{P}(G)$ of G in (a), (c) and (f), respectively

1. For an outer edge $e = \{u, v\}$ in a biconnected plane graph G with $\psi(G) \neq \emptyset$, operation `v-insert`(e) removes the leftmost neighbour $s_1 \in \psi(G)$ of r and inserts a new outer vertex w on the edge $\{u, v\}$, i.e., replaces $\{u, v\}$ with two edges $\{u, w\}$ and $\{w, v\}$. See Fig. 2(a) and (b).
2. For two adjacent outer vertices u and v in a biconnected plane graph G with $\psi(G) \neq \emptyset$, operation `v-add`(u, v) removes the leftmost neighbour $s_1 \in \psi(G)$ of r and adds a new outer vertex w together with two new edges $\{u, w\}$ and $\{w, v\}$. Note that the edge joining u and v is not necessarily an outer edge. See Fig. 2(c) and (d).
3. For two outer vertices u and v which are not adjacent in a biconnected plane graph G, operation `e-add`(u, v) adds a new outer edge $\{u, v\}$. See Fig. 2(e) and (f). Clearly operation `e-add`(u, v) preserves the biconnectivity of G.



(a) G_1 (b) G_2 (c) G_3 (d) G_4

Fig. 3. Biconnected rooted plane graphs. (a) G_1; (b) $G_2 = \mathcal{P}(G_1)$; (c) $G_3 = \mathcal{P}(G_2)$; (d) $G_4 = \mathcal{P}(G_3)$.

Observe that a child G' is a graph obtained from G by one of the operations `e-add`, `v-insert` and `v-add` so that the introduced edge/vertex must be the first removable element of the resulting graph G'. Now we characterize $\mathcal{C}(G)$.

Let u_p (resp., an edge $e_p = \{u_p, u_{p+1}\}$) denote the first removable element when it is a vertex u_p (resp., an edge $e_p = \{u_p, u_{p+1}\}$).

Lemma 3. *Let G_i^1 be the plane graph obtained from G by operation* `v-insert`(e) *for an outer edge $e = \{u = u_i, v = u_{i+1}\}$. Then*

$$G_i^1 \in \mathcal{C}(G) \Leftrightarrow 1 \leq i \leq p-1 \ (\textit{resp.}, \ 1 \leq i \leq p) \ \textit{and} \ \psi(G) \neq \emptyset. \tag{2}$$

Lemma 4. *Let $G_{i,j}^2$ be the plane graph obtained from G by operation* `v-add`(u, v) *for two outer vertices $u = u_i$ and $v = u_j$. Then*

$$G_{i,j}^2 \in \mathcal{C}(G) \Leftrightarrow \psi(G) \neq \emptyset, \{u_i, u_j\} \in E(G), 1 \leq i \leq p, \ \textit{and} \ i+1 \leq j \leq k_{i-1}, \tag{3}$$

except that $G_{1,2}^2 \notin \mathcal{C}(G)$ if $G = F_n$.

Lemma 5. *Let $G_{i,j}^3$ be the plane graph obtained from G by operation* `e-add`(u, v) *for two outer vertices $u = u_i$ and $v = u_j$. Then*

$$G_{i,j}^3 \in \mathcal{C}(G) \Leftrightarrow \{u_i, u_j\} \notin E(G), 1 \leq i \leq p, \ \textit{and} \ i+2 \leq j \leq k_{i-1}. \tag{4}$$

Then $\mathcal{C}(G)$ consists of graphs G_i^1, $G_{i,j}^2$ and $G_{i,j}^3$ in the above lemmas.

5 Algorithm

To generate all plane graphs $G' \in \mathcal{C}(G) \cap \mathcal{G}_2(n, g)$, we generate only those $G' \in \mathcal{C}(G)$ such that the new face introduced by `e-add` or `v-add` and the enlarged face f_i incident to the edge $e_i = \{u_i, u_{i+1}\}$ by `v-insert` are of length at most g.

To generate all biconnected rooted plane graphs in $\mathcal{G}_2(n, g)$, we set $G := F_n$, and execute the following procedure $\textsc{Gen}(G, \varepsilon = u_2)$, where the second argument ε stands for the first removable element in the first argument G. In $\textsc{Gen}(G, \varepsilon)$, we first generate children $G_i^1, G_{i,i+1}^2 \in \mathcal{C}(G)$ for all edges e_i in the active path of G, and then generate children $G_{i,i+\Delta}^2, G_{i,i+\Delta}^3 \in \mathcal{C}(G)$ for all vertices u_i, $i = 1, 2, \ldots, p$ in the active path of G by increasing step size $\Delta \geq 2$ by 1.

Procedure. GEN(G, ε)
Input: A biconnected rooted plane graph $G \in \mathcal{G}_2(n, g)$ and the first removable element ε of G, where ε is either an edge $e_p = \{u_p, u_{p+1}\}$ or a vertex u_p.
Output: All descendants $G' \in \mathcal{G}_2(n, g)$ of G.
begin
/* Let $(u_1 = r, u_2, \ldots, u_B)$ denote the boundary of G in the clockwise order, $e_i = \{u_i, u_{i+1}\}$ denote the edge between u_i and u_{i+1}, and f_i denote the inner face containing e_i. */
if $\psi(G) \neq \emptyset$ **then**
for each edge $e_i = \{u_i, u_{i+1}\}$, $i = 1, 2, \ldots, p$ **do**
if $|f_i| < g$, and "$i < p$" or "$i = p$ and ε is an edge" **then**
Let G' be the graph G_i^1 obtained from G by applying `v-insert`(e_i);
Let w be the newly introduced vertex in e_i; GEN(G', w)
endif;
if $G \neq F_n$ or $i > 1$ **then**
Let G' be the graph $G_{i,i+1}^2$ obtained from G by applying `v-add`(u_i, u_{i+1});
Let w be the newly introduced vertex between u_i and u_{i+1};
GEN(G', w)
endif
endfor
endif;
for $\Delta = 2, 3, \ldots, \min\{B - 1, g - 1\}$ **do**
$i := 1$; $k_0 := +\infty$;
while $i + \Delta \leq k_{i-1}$ and $i \leq p$ **do**
$j := i + \Delta$;
if $\{u_i, u_j\} \notin E(G)$ **then**
Let G' be the graph $G_{i,j}^3$ obtained from G by applying `e-add`(u_i, u_j);
GEN$(G', e_i = \{u_i, u_j\})$
else /* $\{u_i, u_j\} \in E(G)$ */
if $\psi(G) \neq \emptyset$ and $\Delta + 2 \leq g$ **then**
Let G' be the graph $G_{i,j}^2$ obtained from G by applying `v-add`(u_i, u_j);
Let w be the newly introduced vertex between u_i and u_j; GEN(G', w)
endif
endif;
$i := i + 1$;
/* k_{i-1} denotes the index $k \in [i + 1, B]$ of $u_k = \eta_{\mathtt{last}}(e_{i-1})$ */
endwhile
endfor;
Return
end.

Note that the while-loop terminates once it holds $i+\Delta > k_{i-1}$ for some i without executing an iteration for $i' > i$. If $i+\Delta > k_{i-1}$ holds for some i, then it also holds $i' + \Delta > k_{i'-1}$ for any $i' \in [i, p]$ since $k_{i-1} \geq k_{i'-1}$ by (1). Therefore, GEN(G, ε) inspects all possible cases that can generate a child $G' \in \mathcal{C}(G) \cap \mathcal{G}_2(n, g)$.

We first show that each line of GEN(G, ε) can be executed in $O(1)$ time and $O(n)$ space. Since it is easy to maintain data for the size $|f|$ of each inner face f in $O(1)$ per change on an inner face, it suffices to show that $\eta_{\mathtt{last}}(e)$ for each

edge in the active path can be found in $O(1)$ time and that whether two vertices are adjacent or not can be tested in $O(1)$ time.

Lemma 6. *For any edge e in the active path of G, $\eta_{\texttt{last}}(e)$ can be found in $O(1)$ time and $O(n)$ space.*

We test whether given vertices are adjacent or not in $O(1)$ time and $O(n)$ space by using a balanced orientation. Here we show that a balanced orientation always exists in a biconnected plane graph and how to maintain a balanced orientation during an execution of GEN.

Lemma 7. *Every planar graph admits a balanced orientation. A balanced orientation of a biconnected plane graph with n vertices can be maintained in $O(1)$ time per operation for generating a child using $O(n)$ space after an $O(n)$ time preprocessing to the initial input $G := F_n$.*

Finally we show that $\textsc{Gen}(F_n, \varepsilon = u_2)$ can be implemented to run in $O(|\mathcal{G}_2(n,g)|)$ time. For this, it suffices to show that the time complexity $T(G)$ of $\textsc{Gen}(G, \varepsilon)$ without including the computation time for recursive calls of $\textsc{Gen}(G', \varepsilon)$ is $O(|\mathcal{C}(G) \cap \mathcal{G}_2(n,g)|)$. In the first for-loop, G' by $\texttt{v-add}(u_i, u_{i+1})$ is always a child in $\mathcal{C}(G) \cap \mathcal{G}_2(n,g)$ for each e_i (except for the case of $i = 1$ and $G = F_n$). Thus the delay spent to generate the next child G' is $O(1)$ time during the first for-loop. We next show that the delay spent to generate the next child G' is $O(1)$ time during the iteration of the while-loop for each $\Delta \in [2, \min\{B-1, g-1\}]$. If $\psi(G) \neq \emptyset$ and $\Delta + 2 \leq g$ then either $G^3_{i,j}$ or $G^2_{i,j}$ is a child in $\mathcal{C}(G) \cap \mathcal{G}_2(n,g)$, and the delay between two children G' is $O(1)$. Consider the case of $\psi(G) \neq \emptyset$ or $\Delta + 2 > g$. In this case, $G^3_{i,j}$ $(j = i + \Delta)$ is a child in $\mathcal{C}(G) \cap \mathcal{G}_2(n,g)$ only when $\{u_i, u_j\} \notin E(G)$. However, if $\{u_i, u_j\} = \{u_i, u_{i+\Delta}\} \in E(G)$ holds and $G^3_{i,j} \notin \mathcal{C}(G) \cap \mathcal{G}_2(n,g)$ for some i, then $G^3_{i+1,j+1} \in \mathcal{C}(G) \cap \mathcal{G}_2(n,g)$ for the next $i+1$ since $\{u_i, u_j\} = \{u_i, u_{i+\Delta}\} \in E(G)$ implies $\{u_{i+1}, u_{j+1}\} = \{u_{i+1}, u_{i+\Delta+1}\} \notin E(G)$ by the planarity of G.

Hence the delay between two children G' in the while-loop is also $O(1)$ time. This proves that the time complexity $T(G)$ of $\textsc{Gen}(G, \varepsilon)$ without recursive calls $\textsc{Gen}(G', \varepsilon)$ is $O(|\mathcal{C}(G) \cap \mathcal{G}_2(n,g)|)$, and that $\textsc{Gen}(F_n, \varepsilon = u_2)$ runs in $O(|\mathcal{G}_2(n,g)|)$ time. Furthermore, if we output a child G' before calling $\textsc{Gen}(G', \varepsilon)$ when the current depth of recursive call is odd and after calling $\textsc{Gen}(G', \varepsilon)$ when the current depth of recursive call is even, then the delay between two outputs in the entire execution is $O(1)$ in the worst case. It is easy to see that the entire algorithm $\textsc{Gen}(F_n, \varepsilon = u_2)$ can be implemented in $O(n)$ space.

Theorem 1. *For integers $n \geq 4$ and $g \geq 3$, all biconnected rooted plane graphs with exactly n vertices such that each inner face is of length at most g can be enumerated without duplication in $O(n)$ space by an algorithm that outputs the difference between two consecutive outputs in $O(1)$ time in a series of all outputs after an $O(n)$ time preprocessing.*

By defining the parent of F_i, $i > 1$ to be F_{i-1}, we can easily obtain the following corollary.

Corollary 1. *For integers $n \geq 4$ and $g \geq 3$, all biconnected rooted plane graphs with at most n vertices such that each inner face is of length at most g can be enumerated without duplication in $O(n)$ space by an algorithm that outputs the difference between two consecutive outputs in $O(1)$ time in a series of all outputs.*

Corollary 2. *For a given integer $n \geq 4$ and $g \geq 3$, all biconnected plane graphs with exactly n vertices such that each inner face is of length at most g can be enumerated without duplication in $O(n)$ space by an algorithm that outputs the difference between two consecutive outputs in $O(n^3)$ time in average in a series of all outputs.*

6 Concluding Remarks

In this paper, we gave an enumeration algorithm for the class of biconnected rooted plane graphs with exactly n vertices and bounded inner face size g. The algorithm is designed based on graph structures of biconnected plane graphs and an efficient procedure of testing the adjacency of two given vertices. It is our future work to design enumeration algorithms for rooted plane graphs with a higher vertex-connectivity and to take into account the reflectional symmetry around the root, as studied in our companion paper [22].

References

1. Beyer, T., Hedetniemi, S.M.: Constant time generation of rooted trees. SIAM Journal on Computing 9, 706–712 (1980)
2. Fujiwara, H., Wang, J., Zhao, L., Nagamochi, H., Akutsu, T.: Enumerating tree-like chemical graphs with given path frequency. Journal of Chemical Information and Modeling 48, 1345–1357 (2008)
3. Goldberg, L.A.: Efficient Algorithms for Listing Combinatorial Structures. Cambridge University Press, New York (1993)
4. Hall, L.H., Dailey, E.S.: Design of molecules from quantitative structure-activity relationship models. 3. role of higher order path counts: path 3. J. Chem. Inf. Comp. Sci. 33, 598–603 (1993)
5. Hopcroft, J.E., Wong, J.K.: Linear time algorithm for isomorphism of planar graphs. In: STOC 1974, pp. 172–184 (1974)
6. Horváth, T., Ramon, J., Wrobel, S.: Frequent subgraph mining in outerplanar graphs. In: Proc. 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 197–206 (2006)
7. Imada, T., Ota, S., Nagamochi, H., Akutsu, T.: Enumerating stereoisomers of tree structured molecules using dynamic programming. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 14–23. Springer, Heidelberg (2009)
8. Ishida, Y., Zhao, L., Nagamochi, H., Akutsu, T.: Improved algorithm for enumerating tree-like chemical graphs. In: The 19th International Conference on Genome Informatics, Gold Coast, Australia, December 1- 3 (2008); Genome Informatics 21, 53-64 (2008)
9. Kreher, D.L., Stinson, D.R.: Combinatorial Algorithms. CRC Press, Boca Raton (1998)

10. Li, Z., Nakano, S.: Efficient generation of plane triangulations without repetitions. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 433–443. Springer, Heidelberg (2001)
11. Li, G., Ruskey, F.: The advantage of forward thinking in generating rooted and free trees. In: Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 939–940 (1999)
12. Mauser, H., Stahl, M.: Chemical fragment spaces for de novo design. J. Chem. Inf. Comp. Sci. 47, 318–324 (2007)
13. McKay, B.D.: Isomorph-free exhaustive generation. J. of Algorithms 26, 306–324 (1998)
14. Nakano, S.: Efficient generation of plane trees. Information Processing Letters 84, 167–172 (2002)
15. Nakano, S.: Efficient generation of triconnected plane triangulations. Computational Geometry Theory and Applications 27(2), 109–122 (2004)
16. Nakano, S., Uno, T.: Efficient generation of rooted trees, NII Technical Report, NII-2003-005 (2003)
17. Nakano, S., Uno, T.: Generating colored trees. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 249–260. Springer, Heidelberg (2005)
18. Read, R.C.: How to avoid isomorphism search when cataloguing combinatorial configurations. Annals of Discrete Mathematics 2, 107–120 (1978)
19. Wilf, H.S.: Combinatorial Algorithms: An Update. SIAM, Philadelphia (1989)
20. Wright, R.A., Richmond, B., Odlyzko, A., McKay, B.D.: Constant time generation of free trees. SIAM J. Comput. 15, 540–548 (1986)
21. Yamanaka, K., Nakano, S.: Listing all plane graphs. In: Nakano, S.-i., Rahman, M. S. (eds.) WALCOM 2008. LNCS, vol. 4921, pp. 210–221. Springer, Heidelberg (2008)
22. Zhuang, B., Nagamochi, H.: Enumerating rooted biconnected planar graphs with internally triangulated faces, Dept. of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Technical Report 2009-018 (2009), http://www-or.amp.i.kyoto-u.ac.jp/members/nag/Technical report/TR2009-018.pdf
23. Zhuang, B., Nagamochi, H.: Enumerating biconnected rooted plane graphs, Dept. of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Technical Report 2010-001 (2010), http://www-or.amp.i.kyoto-u.ac.jp/members/nag/Technical report/TR2010-001.pdf

Solving General Lattice Puzzles

Gill Barequet[1] and Shahar Tal[2]

[1] Center for Graphics and Geometric Computing
Dept. of Computer Science
Technion—Israel Institute of Technology
Haifa 32000, Israel
barequet@cs.technion.ac.il
[2] Dept. of Computer Science
The Open University
Raanana, Israel
shahar_t@hotmail.com

Abstract. In this paper we describe implementations of two general methods for solving puzzles on *any* structured lattice. We define the puzzle as a graph induced by (finite portion of) the lattice, and apply a back-tracking method for iteratively find all solutions by identifying parts of the puzzle (or transformed versions of them) with subgraphs of the puzzle, such that the entire puzzle graph is covered without overlaps by the graphs of the parts. Alternatively, we reduce the puzzle problem to a submatrix-selection problem, and solve the latter problem by using the "dancing-links" trick of Knuth. A few expediting heuristics are discussed, and experimental results on various lattice puzzles are presented.

Keywords: Polyominoes, polycubes.

1 Introduction

Lattice puzzles intrigued the imagination of "problem solvers" along many generations. Probably the most popular lattice is the two-dimensional orthogonal lattice, in which puzzle parts are so-called "polyominoes" (edge-connected sets of squares), and the puzzle is a container which should be fully covered without overlaps by translated, rotated, and, possibly, also flipped versions of the parts.

One very popular set of parts is the 12 "pentominoes" (5-square polyominoes), which is shown in Figure 1. We describe here a few examples of the many works on pentomino puzzles. Such games were already discussed in the mid 1950's by Golomb [Go54] and Gardner [Ga57]. Scott [Sc58] found all 65 essentially-different (up to symmetries) solutions of the 8×8 puzzle excluding the central 2×2 square. Covering the entire 8×8 square with the twelve pentominoes and one additional 2×2 square part, without insisting on the location of the latter part, is the classic Dudeney's puzzle [Du08]. The Haselgrove couple [HH60], as well as Fletcher [Fl65], computed the 2,339 essentially-different solutions to the 6×10 pentomino puzzle. Other pentomino puzzles were

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 124–135, 2010.

discussed by de Bruijn [Br71]. If pentominoes are not allowed to be flipped upside-down, then there exist 18 such 1-sided pentominoes. Golomb [Go65] provided one tiling of a 9×10 rectangle by all these pentominoes, while Meeus [Me73] credited Leech for finding all the 46 tilings of a 3×30 rectangle by the set of 1-sided pentominoes. Haselgrove [Ha74] found one of the 212 essentially-different tilings of a 15×15 square by 45 copies of the "Y" polyomino. Golomb [Go65] provided many other polyomino puzzles in his seminal book "Polyominoes." See also [Kn00] for a listing of other well-studied puzzles. The Java applet "Jerard's Universal Polyomino Solver"[1] is highly optimized for puzzles on a regular orthogonal lattice, and can supposedly solve any puzzle on this lattice.

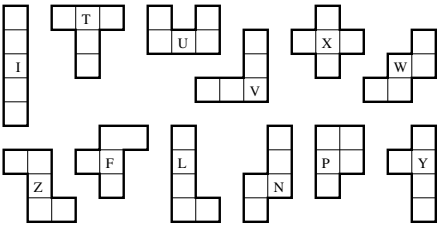


Fig. 1. Twelve essentially-different 2-dimensional pentominoes

It is well-known that finite-puzzle problems on an orthogonal lattice are NP-Complete. This is usually shown by a reduction from the Wang tiling problem, which is also known to be NP-Complete [Le78]. It is not surprising, then, that no better method than back-tracking is used for solving puzzles. Many works, e.g., [Sc58, GB65, Br71], suggested heuristics for speeding-up the process, usually by taking first steps with the least number of branches (possible next steps).

Knuth [Kn00] suggested the *dancing links* method, which allowed solving several problems, including orthogonal and triangular lattice puzzles, much more efficiently than before. The main idea is a combination of a link-handling "trick" that enables easy and efficient "unremove" operations of an object from a doubly-connected list (a key step in back-tracking), and an abstract representation of the problems as a matrix-cover problem: Given a 0/1-matrix, choose a subset of its rows such that each column of the shrunk matrix contains exactly one '1' entry. The reader is referred to the cited reference for more details about this extremely elegant method.

In this paper we present two back-tracking approaches to solving *general lattice* puzzles. We formulate the puzzle problem in general terms, so that it would fit various types of puzzles. Since solving a puzzle even on a two-dimensional orthogonal lattice is NP-Complete, we cannot hope (unless P=NP) for subexponential algorithms for the problem. We fully implemented two algorithms and a few heuristics to expedite them, and ran them on many puzzle problems which lie on several types of lattices.

This paper is organized as follows. In Section 2 we define the puzzle-solving problem. In Section 3 we describe two algorithms to solve it, and present in Section 4 our experimental results. We end in Section 5 with some concluding remarks.

[1] Available at `http://www.xs4all.nl/~gp/Site/Polyomino_Solver.html`

2 Statement of the Problem

Let $\mathcal{L}$ be a *lattice* generated by a repeated pattern. To avoid confusion, let us refer to the dual of $\mathcal{L}$, that is, to its adjacency graph, and use interchangeably the terms "cell" (of the lattice) and "node" (of the graph). A "pattern," in it simplest form, is a single node v and a set of labeled half-edges adjacent to it. (Each undirected edge e of the graph is considered as a pair of half-edges, each of which is outgoing from one endpoint of e.) The half-edges incident to v are labeled 0 through $d(v) - 1$, where $d(v)$ is the degree of v. In addition, there is a specification of how copies of v connect to each other to form $\mathcal{L}$. For a 1-node pattern, this specification is simply a pairing of the labels. In addition, there is a set of transformations defined on v. A transformation is a permutation on the set of labels, i.e., a 1-to-1 mapping between the set $\{0, \ldots, d(v) - 1\}$ and itself.

For example, the orthogonal two-dimensional lattice is generated by the pattern shown in Figure 2: The pattern contains a single node, v, of degree 4. The four half-edges adjacent to v are labeled 'W'=0, 'N'=1, 'E'=2, and 'S'=3, which may be coupled exactly with 'E,' 'S,' 'W,' and 'N,' respectively. The pairing specification is, then, $\{(0, 2), (1, 3)\}$. Only one transformation, so-called *rotate left* (RL, in short) is defined on v. The mapping defined by RL on the half-edges incident to v is $\mathrm{RL}(i) = (i + 1) \bmod 4$. The composition of one to four RL transformations brings v to any possible orientation.

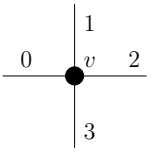


Fig. 2. Repeated pattern of $\mathbb{Z}^2$

Another example is shown in Figure 3: The repeated pattern of this 3D lattice (see Figure 5(c)) is a prism with a hexagonal cross-section. The degree of the single-node pattern v is 8. The eight half-edges, with labels $0, \ldots, 7$, are coupled by $\{(0, 1), (2, 5), (3, 6), (4, 7)\}$. Two transformations are defined on v:
(a) An "x-flip": $F(0|1|2|3|4|5|6|7) = (1|0|2|7|6|5|4|3)$; and
(b) A "yz-rotate": $R(0|1|2|3|4|5|6|7) = (0|1|3|4|5|6|7|2)$.
Compositions of F and R bring v (with repetitions) to all possible orientations in this lattice.

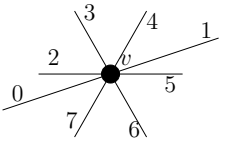


Fig. 3. The hexagonal-prism pattern

A lattice puzzle $\mathcal{P}$ is a finite subgraph of $\mathcal{L}$. Dangling half-edges (that is, half-edges that are *not* coupled with half-edges of other copies of the repeated pattern) are marked as the *boundary* of the puzzle. Puzzle parts are also finite subgraphs of $\mathcal{L}$, and they undergo a similar procedure. A solution to the puzzle is a *covering* of $\mathcal{P}$ by a collection of parts, such that:

1. All nodes of $\mathcal{P}$ are covered by nodes of the parts;
2. The labels of dangling half-edges of parts match the labels of the respective half-edges of $\mathcal{P}$; except
3. Dangling half-edges of parts may extend out of the boundary of $\mathcal{P}$.

Some freedom can be given to the set of parts, so as to form variants of the puzzle problem. We may have a single copy of a given part, or an unlimited amount of copies. Different types of transformations may be defined for different parts of the puzzle.[2] While solving a puzzle, we may want to compute one solution (thus, determine whether or not the puzzle is solvable), or to find the entire set of solutions to the puzzle.

3 Algorithms

In the full version of the paper we illustrate a simple reduction from bin-packing (which is known to be an NP-Complete problem [GJ79]) to puzzle solving, showing that the latter is also NP-Complete. The work [DD07] and references therein provide alternative proofs.

In this section we describe two puzzle-solving algorithms: Direct back-tracking and matrix cover. Since both paradigms are well-known, emphasis is put not on these methods but rather on the data structures and heuristics used to expedite the algorithms in the setting of *general-lattice* puzzles.

3.1 Back-Tracking

Our first approach to solving the puzzle problem is by direct back-tracking.

Puzzle and Part Data Structures. As mentioned above, the puzzle and parts are represented by graphs. Edges of the graphs are labeled with numbers: for each node v, the outgoing half-edges of v are labeled $0, \ldots, d(v) - 1$, where $d(v)$ is the degree of v. Dangling half-edges of both puzzle and parts (that is, half-edges beyond their boundaries), are omitted.

One specific node of each part is marked as the *origin* of the part. For example, in the two-dimensional orthogonal lattice, one may fix the origin at the leftmost cell of the topmost row of the part. Note that the edge labels have to induce a total order on the neighbors of a cell, thereby, on all the cells of the puzzle or its parts. In a general lattice, we simply choose the lexicographically-smallest cell, induced by this order, as the origin of the part. Similarly, we mark the origin of the puzzle. For example, in the two-dimensional orthogonal lattice, a cell is always "smaller" than its bottom and left neighbors, and "greater" than its top and right neighbors. This implies the chosen origin in this lattice.

Part Orientations. A special data structure stores the transformations allowed for each part of the puzzle. As mentioned above, a transformation is a 1-1 mapping between the set of possible labels and itself. In a preprocessing step,

[2] For example, consider the *flip* transformation in a two-dimensional orthogonal-lattice puzzle. Its analogue in three dimensions is the *mirror* transformation, which is mathematically well defined but hard to realize with physical puzzle pieces made of wooden cubes. In such a puzzle, whether or not to allow the mirror transformation is a matter of personal taste.

we compute all the possible orientations of each part. Specifically, we apply all possible transformations (and combinations of them) to the half-edges outgoing from the origin cell. Applying a single combination to the origin changes the orientations of all of its neighbors (manifested by new edge labels), and the process continues recursively in a depth-first manner to all the cells of the part.

Naturally, the graph that represents a part may contain cycles, in which case "back-edges" are encountered in the course of the search. Note that the formal definition of a lattice does *not* ensure that back-edges are consistent edge-labeling-wise. Such an inconsistency simply means that a specific transformation (the analogue of a rotation) is not allowed in the dealt-with lattice. However, in all lattices we experimented with, such a situation can never occur; therefore, we did never check for consistency of back-edges.

After all orientations of a part have been found, two steps should be taken: (a) Recomputing the origin cell of each oriented copy; (b) Removing duplicate copies. In fact, both steps can be performed simultaneously. Moreover, the removal of duplicates may be avoided if we precompute the symmetries of the original part. Nevertheless, all these operations are performed in a preprocessing step, before running the main puzzle-solving algorithm (which takes the main bulk of the running time), so any approach will practically do.

Essentially-Different Solutions. In most cases we are not interested in finding multiple solutions that are inherently the same, up to some transformation defined for the specific lattice. Instead of computing all the solutions and look for repetitions (an operation which might render the algorithm infeasible if there are too many solutions), we applied a simple pruning method. Suppose that S is a solution to a puzzle. One only need to observe that if the entire puzzle has some symmetry realized by the transformation T, then $T(S)$ is also a solution to the puzzle. Thus, if we discard all copies of an arbitrary part, obtained by transformations that realize symmetries of the puzzle, we guarantee that only essentially-different solutions to the puzzle will be found. To maximize efficiency, we choose the part with the maximum number of copies. (Usually, this is the part with the least number of symmetries.)

Solving the Puzzle. Our first method is a classical back-tracking algorithm. We attempt systematically to *cover* the puzzle graph with the part graphs. A part graph covers exactly a portion of the puzzle graph by an injective mapping of the nodes of the part to the nodes of the puzzle, subject to adjacency relations in the two graphs, while the labels of the (half-)edges of the part fully match those of the covered portion in the puzzle.

Initially, all parts are "free," and all nodes in the puzzle graph are "empty." We initialize a variable, called the *anchor*, to be the origin cell of the puzzle. Naturally, it should be covered by some cell c of some part p. Moreover, c must be the origin of p, otherwise, after positioning p in the puzzle, the origin of p will occupy a puzzle cell which is different from the origin of the puzzle, which is a contradiction to the lexicographic minimality of the puzzle origin. Thus, positioning a part in the puzzle (in one of its possible orientations) is achieved by identifying the anchor with the origin of the part and traversing the part

graph. A simultaneous and identical (edge-label-wise) traversal is performed in the puzzle graph. This yields precisely which portion of the puzzle is covered by the part. An attempt to position a part in the puzzle fails if either the traversal of the puzzle graph reaches an "occupied" cell, or it gets out of the graph (that it, the boundary of the puzzle is about to be crossed).

If the part-positioning is successful, we mark the part as "used," mark all the nodes of the puzzle graph that are matched to nodes of the positioned part as "occupied," and proceed to the next part-positioning step as follows. First, we set the anchor of the puzzle to be the new lexicographically-minimal "empty" node in the puzzle graph. This is achieved by scanning the puzzle lexicographically, starting from the previous anchor and looking for a free node. Then, we attempt to position a new "free" part (according to a predefined order) in the puzzle by identifying its origin with the new anchor and proceeding as above. The new anchor must be the origin of the next positioned part for the same reason as for the first positioned part.

This part-positioning process continues until one of two things happen: Either (a) The puzzle graph is fully covered (this situation is identified by not being able to reset the anchor); or (b) The algorithm is stuck in a situation in which the puzzle graph is not fully covered, yet no new part can be positioned. In the former situation the algorithm declares a solution and terminates. In the latter situation the algorithm back-tracks: The last positioned part is removed from the puzzle, restoring its status to "free" and marking again the covered puzzle nodes as "empty." Then, if another orientation of the same part exists, the algorithm attempts to position it as above. Otherwise, the algorithm proceeds to the next available part (according to the predefined order) and attempts to position it in the same manner.

In case we like to find *all* solutions to the puzzle, a minor step of the algorithm is modified: When the algorithm finds a solution, the latter is reported, but then refers to the last part-positioning step as a failure and then back-tracks. In such a situation, the algorithm terminates when no back-tracking options remain: This happens when all options for the *first* part-positioning are exhausted.

We conclude the description of the algorithm by referring to the dangling half-edges in part graphs. In principle, they should be matched too to half-edges in the puzzle graph, in order to ensure proper *neighborhoods* between parts positioned in the puzzle. However, this was redundant in all the lattices that we handled. Thus, we ignored all dangling half-edges in the part-positioning operations. A cover of the puzzle graph was actually only an exact cover of the *nodes* of the graph. Edges in the puzzle graph, that represent neighborhood relations between parts, were not covered. Exact match of labels was enforced only between *inter-part* edges and the corresponding edges in the puzzle graph.

Stranding. A simple pruning method is to consider the size of the connected component of empty nodes, that include the anchor, in the puzzle graph. If all free parts are larger than this component, then the algorithm may back-track without any further checks. This heuristic can also be applied for the second algorithm described below.

3.2 Matrix Cover

Our second approach to solving the puzzle problem is by a reduction to *matrix cover*, and speeding up the algorithm that solves the latter problem by using the "dancing links" technique [Kn00].

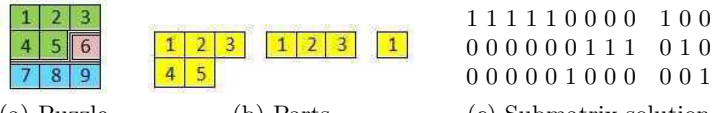
Reduction. The puzzle problem is represented by a binary matrix in the following manner. We create an $M \times N$ matrix, where M is the total number of options to position parts (in all orientations) in the puzzle, and N is the number of cells of the graph. By "part-positioning" options we mean all possible mappings of the anchor of a part graph to a node in the puzzle graph, such that the part will partially cover the puzzle and will not exceed its boundary. An entry (i, j) in the matrix contains the value 1 if the jth node of the puzzle is covered by some node in the ith part-positioning option; otherwise it is 0.

Naturally, solving the puzzle amounts to choosing a subset of the rows in which every column contains a single 1 with 0 in all other entries. The chosen rows represent the choice of parts, while the requirement for a single 1 per column guarantees that every cell of the puzzle will be covered by exactly one part.

We *may* want to ensure that every part will be used exactly once, among all its possible orientations and positions in the puzzle. This is easily achieved by adding more columns to the matrix, one for each part. In each such column, we put 1 in all the rows that correspond to the same part, and 0 elsewhere. This guarantees that exactly one part orientation and position is chosen.

For example, consider the simple 3X3 puzzle shown in Figure 4(a). The three puzzle parts are shown in Figure 4(b). The left part has eight different orientations (allowing flipping), in each of which it can be positioned in the puzzle in two ways, for a total of 16 options. The middle part has two different orientations, in each of which it can be positioned in the puzzle in three ways, for a total of 6 options. Finally, the right part has only one orientation, which can be positioned in the puzzle in nine ways. Thus, the matrix we need to cover is made of 31 rows (the total number of part-positioning options) and 12 columns (9 puzzle cells plus three original parts). One possible solution to the puzzle is represented by the 3-row submatrix shown in Figure 4(c): The 9 left columns show the exact covering of the puzzle, while the 3 right columns show that each part is used exactly one.

The matrix-cover algorithm is also back-tracking in nature, and so its details are not provided here. However, its running time is reduced significantly as described below.



(a) Puzzle (b) Parts (c) Submatrix solution

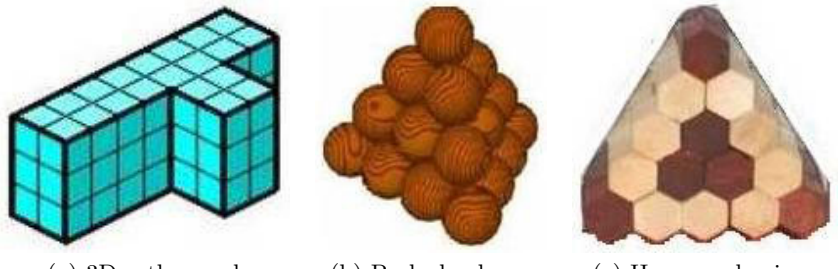
Fig. 4. A puzzle solution represented by a submatrix

Dancing Links. The algorithm can be sped up tremendously by an elegant pointer-manipulation trick [Kn00]. In a doubly-connected linked list, an element x is removed by executing the two operations `Next[Prev[`x`]]` := `Next[`x`]` and `Prev[Next[`x`]]` := `Prev[`x`]`. It is a good programming practice not to access the storage of an element x after it is deleted, since it may already serve a different purpose. However, if it is guaranteed that the area allocated to x is not altered even after it is deleted, one can easily undo the deletion of x by making the links "dance": The pair of operations `Next[Prev[`x`]]` := x and `Prev[Next[`x`]]` := x readily return x to its original location in the linked list.

Efficient Branching (Size Heuristic). A very efficient heuristic, which reduces the running time significantly, is ordering the branches of the algorithm according to their (anticipated) size. The speed-up of the algorithm naturally depends on the quality of the prediction of the sizes of branches. In our setting, branching occurs at the selection of an additional column. Recall that in a valid solution, exactly one of the rows comprising the submatrix contains the value 1 in this column. (This means that exactly one part covers the puzzle cell corresponding to this column.) Since no a priori information is known (or computed), a reasonable choice is to explore first columns with the *least* number of 1-entries. This is because we will have fewer parts among which to choose the one covering this cell of the puzzle.

4 Experimental Results

The two algorithms were implemented in Java on a dual-core 2.2GHz PC with Sun's Virtual Machine 5+, under the MS Windows XP and Linux operating systems. The two algorithms allowed simple parallelism for using the dual-core CPU by splitting the search tree into two branches. All the running times reported below were measured on MS Windows XP without parallelism. (Using the dual-core CPU reduced the running time by half in practically all cases.) Each puzzle was solved twice (with a random order of parts), and the reported value for each puzzle is the average of the two measurements. The software consists of about 10,000 lines of code. In addition, we implemented a feature which allowed



(a) 3D orthogonal (b) Packed spheres (c) Hexagonal prisms

Fig. 5. Lattice puzzles

the user to save the current search state in the direct back-tracking method and to reload it later for a warm-restart of the program.

The first two lattice types which we experimented with were the two- and three-dimensional orthogonal lattices. The formal structure of the 2D lattice is described in the introduction of this paper. It is straightforward to generalize it to

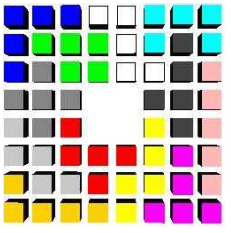
Table 1. Statistics of puzzle solving

	Puzzle		Parts				Solutions		Method &	Branching	Time (Sec.)	
Lattice	Name	Size	Orig.	Total	Reduced	Poses	Unique	Sym.	Heuristic(s)	Points	First	All
2D Ort.	$8 \times 8 - 2 \times 2$	60	12	56	56	1,290	65	*8	BT	2,757,203	0.2	5.7
									BT+ST	1,523,328	0.5	3.5
									MC	2,910,535	0.2	12.4
									MC+ST	2,383,728	0.3	9.8
									MC+SZ	117,723	0.2	1.1
									MC+SZ+ST	81,880	0.1	0.8
2D Ort.	10×6	60	12	57	57	1,758	2,339	*4	BT	10,595,621	0.1	17.9
									BT+ST	5,802,074	0.1	14.4
									MC	10,606,305	0.2	54.4
									MC+ST	9,242,529	0.1	43.3
									MC+SZ	1,732,537	0.2	11.8
									MC+SZ+ST	1,480,505	0.1	11.2
3D Ort.	$10 \times 3 \times 2$	60	12	168	68	1,416	12	*8	BT	1,315,930	0.2	2.3
									BT+ST	861,525	0.1	2.0
									MC(+ST)	1,325,957	0.3	6.3
									MC+SZ(+ST)	74,947	0.2	0.7
3D Ort.	$5 \times 4 \times 3$	60	12	168	142	2,168	3,940	*8	BT	1,969,089,192	0.4	5,191
									BT+ST	1,259,189,714	1.1	4,165
									MC	1,969,152,287	0.6	5,874
									MC+ST	1,969,152,287	0.8	5,912
									MC+SZ	10,107,229	0.2	68
									MC+SZ+ST	10,108,090	0.3	65
3D Ort.	$6 \times 5 \times 2$	60	12	168	93	1,916	264	*8	BT	107,590,605	4.8	223
									BT+ST	67,714,344	4.2	190
									MC	107,620,901	1.9	395
									MC+ST	107,620,901	0.8	421
									MC+SZ	759,343	0.2	7.2
									MC+SZ+ST	693,970	0.4	6.7
3D Ort.	Green Happy Cube	98	6	91	91	182	20	*24	BT	1,262	0.2	0.2
									BT+ST	1,215	0.1	0.2
									MC(+ST)	3,318	0.2	0.3
									MC+SZ(+ST)	234	0.0	0.2
3D Ort.	Orange Happy Cube	98	6	79	79	158	2	*24	BT	863	0.1	0.4
									BT+ST	846	0.1	0.3
									MC(+ST)	2,235	0.1	0.3
									MC+SZ(+ST)	146	0.0	0.3
3D Ort.	Strip	60	12	168	85	965	6	*4	BT	187,883	0.0	0.7
									BT+ST	99,272	0.2	0.6
									MC	203,197	0.3	1.7
									MC+ST	188,506	0.1	1.6
									MC+SZ	14,274	0.0	0.4
									MC+SZ+ST	13,904	0.0	0.3
3D Ort.	Stairs	55	12[a]	186	186	1,573	640	*1	BT	3,143,814	0.1	12.9
									BT+ST	2,088,970	0.1	10.8
									MC	3,143,784	0.1	23
									MC+ST	3,100,112	0.1	18.8
									MC+SZ	178,350	–[a]	1.8
									MC+SZ+ST	159,868	–[a]	1.4
3D Ort.	Big Y	60	12	186	157	1,640	14	*1	BT	210,454,691	52	536
									BT+ST	161,584,456	26	513
									MC	210,502,803	29	529
									MC+ST	210,502,803	2.8	560
									MC+SZ	205,230	0.3	1.6
									MC+SZ+ST	205,524	0.3	1.6
Spheres	Pyramid	20	6	52	31	85	1	*12	BT+(ST)	634	0.0	0.1
									MC+(ST)	1,994	0.1	0.2
									MC+SZ(+ST)	171	0.1	0.1
Hex Prism	Hex prisms	45	11	104	98	232	2	*6	BT	4,675	0.0	0.2
									BT+ST	4,029	0.0	0.2
									MC+(ST)	5,902	0.1	0.2
									MC+SZ(+ST)	422	0.1	0.2

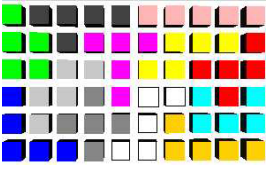
Legend: BT - back tracking; MC - matrix-cover (with dancing links); ST - stranding heuristic; SZ - size heuristic.

[a] In this puzzle one (a priori unknown) part was redundant. This caused the SZ heuristic to not be able to find any solution.

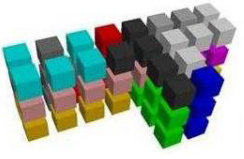
three dimensions. Figure 5(a) shows a sample puzzle on the 3D lattice. The third lattice type is the "packed-spheres" lattice. Figure 5(b) shows a sample puzzle on this lattice. In this lattice, the repeated pattern is a node of degree 12. A complete characterization of the group of transformations in this lattice will be provided in the full version of the paper. The fourth lattice type is the "hexagonal-prism" lattice, whose structure is also described in the introduction. Figure 5(c) shows a sample puzzle on this lattice.



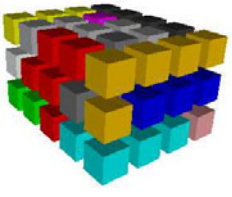
(a) 2D $8 \times 8 - 2 \times 2$



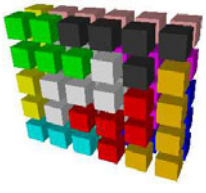
(b) 2D 10×6



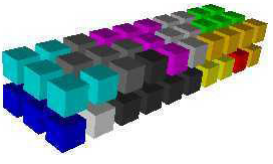
(c) 3D Big Y



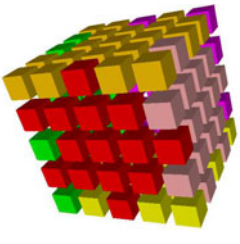
(d) 3D $5 \times 4 \times 3$



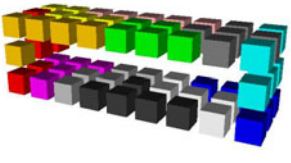
(e) 3D $6 \times 5 \times 2$



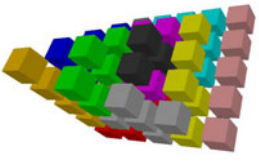
(f) 3D $10 \times 3 \times 2$



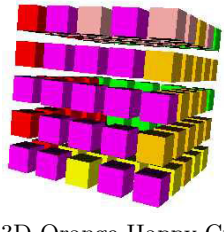
(g) 3D Green Happy Cube



(h) 3D Strip

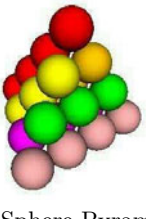


(i) 3D Stairs



(j) 3D Orange Happy Cube

(k) Hexagonal Prisms



(l) Sphere Pyramid

Fig. 6. Solutions to various puzzles

We experimented with many puzzles on these lattices, and report here (see Table 1) on only a few of these puzzles. The *size* of a puzzle is the number of cells it contains. For parts we provide four numbers: the number of original parts, the total number of different oriented parts, the number of oriented parts that can fit into the puzzle, and the total number of options to position oriented parts in the puzzle. (The latter is a good measure of the complexity of a puzzle.) We provide two counts of solutions: essentially different and the total number of solutions. The acronyms BT and MC stand for the direct back-tracking and matrix cover methods, respectively. The ST (stranding) heuristic can be applied to both methods, while SZ (the size heuristic) is relevant to matrix cover only. The notation "X(+Y)" means that applying the heuristic "Y" did not improve the running time relative to using only the method "X." *Branches* are decision points in which the back-tracking algorithm places a part or the matrix-cover algorithm chooses a column. Finally, we report the times needed to find (on MS Windows) either a single solution or all solutions to the puzzle. Figure 6 shows representative solutions to these puzzles.

As can be easily observed from the data in Table 1, the matrix-cover algorithm, coupled with the dancing-links "trick" and using the branch size-ordering heuristic, is superior to the direct back-tracking algorithm. (Due to the inherent representation of the problem in the two algorithms, we do not have any heuristic for the latter algorithm that is similar to the size heuristic for the former algorithm.) The stranding heuristic can modestly improve both algorithms, but it doesn't change the superiority of the matrix-cover algorithm.

5 Conclusion

We present two puzzle-solving algorithms and identify one of them as the method of choice. We experimented with many puzzles in different lattices. Our future goals are to extend the algorithms to other lattices, add more restrictions to the puzzles (e.g., attaching knobs and holes to the parts), implement "SZ" for the back-tracking method, and improve the efficiency of our implementation.

Acknowledgment

The authors wish to thank an anonymous reviewer and Rudolf Fleischer for many valuable comments on the paper. Part of this research was performed while the first author was on sabbatical at Tufts University, Medford, MA.

References

[Br71] De Bruijn, N.G.: Programmeren van de pentomino puzzle. Euclides 47, 90–104 (1971–1972)

[DD07] Demaine, E.D., Demaine, M.L.: Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. Graphs and Combinatorics 23(suppl.), 195–208 (2007)

[Du08] Dudeney, H.E.: 74.—The broken chessboard. In: The Canterbury Puzzles, 90–92 (1908)
[Fl65] Fletcher, J.G.: A program to solve the pentomino problem by the recursive use of macros. Comm. of the ACM 8, 621–623 (1965)
[Ga57] Gardner, M.: Mathematical games: More about complex dominoes, plus the answers to last month;s puzzles. Scientific American 197, 126–140 (1957)
[GJ79] Garey, M., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
[Go54] Golomb, S.W.: Checkerboards and polyominoes. American Mathematical Monthly 61, 675–682 (1954)
[Go65] Golomb, S.W.: Polyominoes, Scribners, New York (1965); 2nd edn. Princeton University Press, Princeton (1994)
[GB65] Golomb, S.W., Baumart, L.D.: Backtrack programming, *J.* of the ACM 12, 516–524 (1965)
[HH60] Haselgrove, C.B., Haselgrove, J.: A computer program for pentominoes. Eureka 23, 16–18 (1960)
[Ha74] Haselgrove, J.: Packing a square with Y-pentominoes. J. of Recreational Mathematics 7, 229 (1974)
[Kn00] Knuth, D.E.: Dancing links. In: Davies, J., Roscoe, B., Woodcock, J. (eds.) Millennial Perspectives in Computer Science, pp. 187–214. Palgrave Macmillan, England (2000), http://arxiv.org/abs/cs/0011047
[Le78] Lewis, H.R.: Complexity of solvable cases of the decision problem for the predicate calculus. In: 19th Ann. Symp. on Foundations of Computer Science, Ann Arbor, MI, pp. 35–47 (1978)
[Me73] Meeus, J.: Some polyomino and polyamond problems. J. of Recreational Mathematics 6, 215–220 (1973)
[Sc58] Scott, D.S.: Programming a combinatorial puzzle, Technical Report 1, Dept. of Electrical Engineering, Princeton University (June 1958)

A Hybrid Graph Representation for Recursive Backtracking Algorithms

Faisal N. Abu-Khzam[1,⋆], Michael A. Langston[2],
Amer E. Mouawad[1], and Clinton P. Nolan[2,⋆⋆]

[1] Department of Computer Science and Mathematics
Lebanese American University
Beirut, Lebanon
{faisal.abukhzam,amer.mouawad}@lau.edu.lb
http://www.csm.lau.edu.lb/fabukhzam
[2] Department of Electrical Engineering and Computer Science
University of Tennessee
Knoxville, TN 37996-3450 USA
{langston,nolan}@eecs.utk.edu
http://www.cs.utk.edu/~langston

Abstract. Many exact algorithms for $\mathcal{NP}$-hard graph problems adopt the old Davis-Putman branch-and-reduce paradigm. The performance of these algorithms often suffers from the increasing number of graph modifications, such as deletions, that reduce the problem instance and have to be "taken back" frequently during the search process. The use of efficient data structures is necessary for fast graph modification modules as well as fast take-back procedures. In this paper, we investigate practical implementation-based aspects of exact algorithms by providing a hybrid graph representation that addresses the take-back challenge and combines the advantage of $\mathcal{O}(1)$ adjacency-queries in adjacency-matrices with the advantage of efficient neighborhood traversal in adjacency-lists.

Keywords: data structures, exact algorithms, recursive backtracking, vertex cover, dominating set.

1 Introduction

The interest in exact algorithms for hard graph problems is now more than fifty years old. Countless exact and parameterized algorithms with improved worst-case run-times have been developed in the last several years only [5,6,11]. However, most research in this area concentrates on worst-case analysis, dealing with data structure design and implementation from an abstract level only.

⋆ This research has been supported in part by the research council of the Lebanese American University.

⋆⋆ This research has been supported in part by the U.S. Department of Energy under the EPSCoR Laboratory Partnership Program. It has used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science under Contract No. DE-AC02-05CH11231.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 136–147, 2010.

In this paper, we investigate the practical aspects of some exact algorithms for $\mathcal{NP}$-hard graph-theoretic problems. The majority of such exact algorithms adopt the classical recursive backtracking methodology, with the adjacency-list or adjacency-matrix as default graph representation. We present a hybrid graph representation that trades space for time to combine the advantage of $\mathcal{O}(1)$ adjacency-queries in adjacency-matrices and the advantage of efficient neighborhood traversal in adjacency-lists.

Our interest in practical and efficient exact implementations of graph algorithms for hard computational problems has been motivated by several recent developments such as:

- More random access memory and cache memory availability making memory-intensive computations more affordable.
- The increased availability of high performance platforms.
- The fact that many $\mathcal{NP}$-complete problems are hard to approximate within reasonable relative errors A notorious example is the Maximum Clique problem [3,9].
- The advances in designing exact and parameterized algorithms, and the fact that search-tree based algorithms are the dominant method.
- Highly demanding application domains. In some application domains, data takes months and years to collect and exact solutions are badly needed. Hence, users are often tolerant to accurate answers that take reasonable time (hours or days).
- Multi-fold approximations. Double inaccuracy arises when "approximate" solutions are provided for "simplified" models of real problems (protein folding methods are typical examples).

This paper is concerned mainly with implementation strategies that are suitable for recursive backtracking algorithms on graphs. Many aspects of our methods can also be used with hyper-graphs and with other types of problems.

2 Background

In the area of exact algorithms and parameterized complexity, the worst-case run-times for many different graph algorithms are constantly being improved. Such improvements usually involve an increase in the number of polynomial-time reductions during search. Due to the exponential number of search-tree nodes, polynomial time (housekeeping) reductions could have a butterfly effect on the efficiency of such algorithms. Obviously, this could render worst-case algorithms less practical then some simpler exact algorithms that tend to require less work at every search-tree node[1].

Moreover, search-tree based algorithms suffer from the increasing number of actions associated with branching decisions that have to be taken, then (frequently) taken back, at every search-tree node. A challenging task, therefore,

[1] We use the expression "worst-case algorithm" when referring to the algorithm (for a given problem) with the current smallest asymptotic upper bound on its run-time.

is to reduce the additional cost of *undo* operations. Generally, every operation is pushed onto a stack and later popped out and performed in reverse. We denote this action by "explicit-undo." Our hybrid graph representation addresses those challenges by reducing the cost of deletion operations via "implicit-undo." To illustrate the efficiency of our representation, several implementations using different techniques were developed and compared for two well-known graph problems: Dominating Set and Vertex Cover.

2.1 Classical Graph Representation

For the sake of completeness, we ought to mention some elementary facts. Graphs are usually represented using one of two data structures: adjacency matrices (AM) or adjacency lists (AL). In addition, we often use a degrees' array to keep track of active vertices and the current cardinalities of their neighborhoods. When using AM, neighborhood traversal takes $\Omega(n)$ where n is the number of vertices in the graph. This is reduced to $\mathcal{O}(d)$, where d is the maximum vertex degree, if we use AL instead. On the other hand, checking if two vertices are adjacent requires $\mathcal{O}(d)$ in AL and $\mathcal{O}(1)$ in AM.

2.2 The Dominating Set Problem

In the Dominating Set problem, henceforth DS, we are given an n-vertex graph $G = (V, E)$, and we are asked to find a set $D \subset V$ of smallest possible cardinality such that every vertex of G is either in D or adjacent to some vertex in D. DS has received great attention, being a classical $\mathcal{NP}$-hard graph optimization problem with many logistical applications.

Until 2004, the best algorithm for DS was still the trivial $\mathcal{O}^*(2^n)$ enumeration[2]. In that same year, three algorithm were independently published breaking the $\mathcal{O}^*(2^n)$ barrier [7,8,10]. The best worst-case algorithm was presented by Grandoni with a running time in $\mathcal{O}^*(1.8019^n)$ [8]. Using measure-and-conquer, a bound of $\mathcal{O}^*(1.5137^n)$ was obtained on the running time of Grandoni's algorithm [4]. This was later improved to $\mathcal{O}^*(1.5063)$ in [13] and the current best worst-case algorithm can be found in [12] where a general algorithm for counting minimum dominating sets in $\mathcal{O}^*(1.5048)$ is also presented.

For our experimental work, we implemented two versions of the algorithm of [4] where DS is solved by reduction to Minimum Set Cover:

- AL_DS_OPT: optimization version using the adjacency-lists representation;
- HYBRID_DS_OPT: optimization version using the hybrid graph representation.

2.3 The Vertex Cover Problem

In the (parameterized) Vertex Cover problem, or VC for short, we are given a graph $G = (V, E)$, together with a parameter k, and we are asked to find a set C

[2] Throughout this paper we use the modified big-Oh notation that suppresses all polynomially bounded factors. For functions f and g we say $f(n) \in \mathcal{O}^*(g(n))$ if $f(n) \in \mathcal{O}(g(n)poly(n))$, where $poly(n)$ is a polynomial.

of cardinality k such that $C \subseteq V$ and the subgraph induced by $V \setminus C$ is edgeless. The current fastest worst-case VC algorithm runs in $\mathcal{O}(kn + 1.2852^k)$ time [1]. An optimization algorithm for VC can be obtained by obvious modifications to the parameterized algorithm of [1], or using the Maximum Independent Set algorithm from [6].

For comparison purposes, four versions were implemented for VC:

- AL_VC_OPT: an optimization version using the adjacency-lists representation, based on simple modifications of the parameterized VC algorithm;
- HYBRID_VC_OPT: an optimization version using the hybrid graph representation;
- HYBRID_VC_PARM: a parameterized version using the hybrid graph representation but not taking advantage of the folding technique described in [1,2];
- HYBRID_VCF_PARM: a parameterized version using the hybrid graph representation and modified for fast edge-contraction operations.

3 The Hybrid Graph Representation

We describe our hybrid graph representation using the example of Figure 1. The adjacency list of a vertex v is stored in an array denoted by AL$[v]$. Accordingly, AL$[v][i]$ holds the index of the i^{th} vertex in the list of neighbors of v. The adjacency matrix, denoted henceforth by IM, is used as an index table for the adjacency list, as follows: the entry IM$[u][v]$ is equal to the index of u in AL$[v]$. IM$[u][v]$ is -1 when the two vertices are not connected.

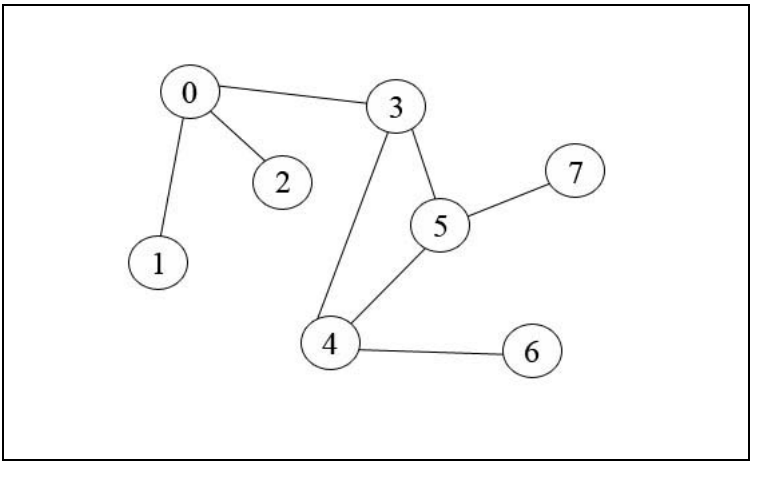


Fig. 1. Graph G

Considering the graph G of Figure 1, the initial contents of AL and IM are as follows:

IM

	0	1	2	3	4	5	6	7
0	−1	0	0	0	−1	−1	−1	−1
1	0	−1	−1	−1	−1	−1	−1	−1
2	1	−1	−1	−1	−1	−1	−1	−1
3	2	−1	−1	−1	0	0	−1	−1
4	−1	−1	−1	1	−1	1	0	−1
5	−1	−1	−1	2	1	−1	−1	0
6	−1	−1	−1	−1	2	−1	−1	−1
7	−1	−1	−1	−1	−1	2	−1	−1

DEGREE

0	3
1	1
2	1
3	3
4	3
5	3
6	1
7	1

AL

	0	1	2
0	1	2	3
1	0		
2	0		
3	0	4	5
4	3	5	6
5	3	4	7
6	4		
7	5		

Note that AL is implemented using a two dimensional array for fast (direct) access via the indexing provided by IM. We allocate enough memory to fit the neighbors of each vertex only. In addition to IM and AL, we introduce three linear arrays: the degree vector (DEGREE), the vertex list (LIST), and the vertex index list (IDXLIST). The degree vector is the current neighborhood cardinality, the vertex list is the list of currently active (not deleted) vertices that will be used instead of the degree vector for more efficient complete graph traversals, and the vertex index list is the index of each vertex in the vertex list. In other words, LIST$[i]$ is the i^{th} vertex in the list of active vertices and IDXLIST$[u]$ is the index of vertex u in LIST.

All data structures except for the DEGREE vector are global and their memory is allocated at startup only. The DEGREE vector is local to every search-tree node (i.e: every search-tree node receives a new copy of the vector).

LIST

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

IDXLIST

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

In the next section, we show how these structures are dynamically modified during the search, while performing various operations. We note that some operations, like edge contraction for example, require additional bookkeeping that we briefly describe later. However, most common operations can be performed using the five data structures described above, which when combined together form the (generic) hybrid graph representation.

4 Efficient Reduction Operations

4.1 Edge Deletion

The simplest and most frequent operation performed during the search is probably edge deletion. Maintaining the hybrid data structure in this case is straightforward. For deleting an edge (u, v), the degrees of u and v are decremented by one and the adjacency lists of the two vertices are adjusted respectively by placing u at the last position of AL$[v]$ and v at the last position of AL$[u]$. Simply, each of these two operations consists of a a single swap with the last element of the respective list, together with an adjustment of the positions in IM.

The delete_edge function

Input: Edge (u,v).

Begin

int i, j, x;
$i = \text{IM}[v][u]$;
$j = --\text{DEGREE}[u]$;
$x = \text{AL}[u][j]$;
$\text{AL}[u][i] = x$;
$\text{AL}[u][j] = v$;
$\text{IM}[x][u] = i$;
$\text{IM}[v][u] = j$;
Repeat the previous steps for $i = \text{IM}[u][v]$ and $j = --\text{DEGREE}[v]$;

End

Going back to our illustrative graph G, after deleting edge (0, 3), the modified AL, IM, and DEGREE will look as follows (changes in bold):

IM

	0	1	2	3	4	5	6	7
0	−1	0	0	**2**	−1	−1	−1	−1
1	0	−1	−1	−1	−1	−1	−1	−1
2	1	−1	−1	−1	−1	−1	−1	−1
3	2	−1	−1	−1	0	0	−1	−1
4	−1	−1	−1	1	−1	1	0	−1
5	−1	−1	−1	**0**	1	−1	−1	0
6	−1	−1	−1	−1	2	−1	−1	−1
7	−1	−1	−1	−1	−1	2	−1	−1

DEGREE

0	**2**
1	1
2	1
3	**2**
4	3
5	3
6	1
7	1

AL

	0	1	2
0	1	2	3
1	0		
2	0		
3	**5**	4	**0**
4	3	5	6
5	3	4	7
6	4		
7	5		

Notice that edge deletion runs in $\mathcal{O}(1)$ since all the information required for switching positions in AL can be found in IM. Now assume we want to undo this operation. This can be accomplished by simply setting $\text{DEGREE}[0] = \text{DEGREE}[3] = 3$. The only difference between this state and the initial state is that the positions of the neighbors have changed in the adjacency lists. Knowing that every search-tree node will maintain its own copy of the DEGREE vector, no actions whatsoever need to be taken to undo this operation, thus we have an "implicit-undo" for edge deletion operations.

Remark 1. Checking if two vertices u and v are adjacent still takes constant time but requires one additional checking: u and v are adjacent when $-1 < \text{IM}[u][v] < \text{DEGREE}[v]$.

4.2 Vertex Deletion

Deleting a vertex v runs in $\Omega(d)$ time where d is the (current) degree of v. We run the edge deletion operation for every active neighbor of v. In addition, we remove

v from the list of active vertices by swapping it with the last active vertex in LIST and decrementing the number of active vertices by one. The IDXLIST plays the same role as IM for deleting a vertex from LIST. As in the case of edge-deletion, the undo of vertex deletion requires no actions since every edge deletion can be considered as an independent operation.

To illustrate the purpose of the LIST vector, consider the two operations of copying the DEGREE vector and searching for the vertex of highest degree. Doing so would consume $\Omega(n)$ time if the DEGREE vector is used alone. This is reduced to $\Omega(n_c)$, where n_c is the number of currently active vertices, when combining the DEGREE and LIST vectors. Iterating from $i = 0$ to n_c, DEGREE[LIST[i]] returns the degree of the vertex at position i in LIST.

delete_vertex function

Input: Vertex v, and total number of active vertices n_c.

Begin

```
int last, i, u, d;
d = DEGREE[v];
last = LIST[n − 1];
i = IDXLIST[v];
LIST[i] = last;
LIST[n − 1] = v;
IDXLIST[last] = i;
IDXLIST[v] = n − 1;
for(i = d − 1; i ≥ 0; i − −)
    u = AL[v][i];
    delete_edge(u, v);
```

End

4.3 Edge Contraction

The next operation we consider is edge contraction. Contracting edge (u, v) replaces vertices u and v by a new vertex whose neighborhood is $N(u) \cup N(v) \setminus \{u, v\}$.

To implement this operation, we use a coloring technique that requires additional bookkeeping. Simply, vertices with the same color are treated as one single vertex obtained by contracting edges between them. Initially, every vertex v_i is assigned color c_i, and every color class c_i has initial cardinality one and degree $d(c_i) = d(v_i)$. In addition to previously discussed data structures, we use the following:

- the VCOLOR vector: holds the current color of every vertex;
- the COLOR_CARD (CC) vector indicates the current cardinality of every color set;
- the COLOR_DEGREE (CD) vector holds the current degree of every color set;

– the COLOR_SET_LIST (CSL) holds the list of vertices belonging to every color set.

The LIST and IDXLIST do not hold vertex information anymore, but they maintain the list of active (not deleted) colors instead, since all operations now involve color sets. When no edge contraction operations are performed, color sets would be identified with their corresponding vertices.

	CD		CC		CSL		DEGREE		AL
					0 1 2				0 1 2
0	3	0	1	0	0	0	3	0	1 2 3
1	1	1	1	1	1	1	1	1	0
2	1	2	1	2	2	2	1	2	0
3	3	3	1	3	3	3	3	3	0 4 5
4	3	4	1	4	4	4	3	4	3 5 6
5	3	5	1	5	5	5	3	5	3 4 7
6	1	6	1	6	6	6	1	6	4
7	1	7	1	7	7	7	1	7	5

VCOLOR	LIST	IDXLIST
0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
[0 1 2 3 4 5 6 7]	[0 1 2 3 4 5 6 7]	[0 1 2 3 4 5 6 7]

When edge contraction is possible, the initial state for the graph G consists of all structures previously shown. AL, IM, CSL, LIST and IDXLIST would be globally stored (in RAM), while DEGREE, CD, CC and VCOLOR would be copied at every search-tree node. To contract an edge (v_0, v_3), we actually assign both vertices the same color. Assuming we assign the two vertices color c_0, the modifications required are shown below in bold (changes to IM not shown here but are required):

	CD		CC		CSL		DEGREE		AL
					0 1 2				0 1 2
0	**4**	0	**2**	0	0 **3**	0	**2**	0	1 2 3
1	1	1	1	1	1	1	1	1	0
2	1	2	1	2	2	2	1	2	0
3	**0**	3	**0**	3	3	3	**2**	3	**5** 4 **0**
4	3	4	1	4	4	4	3	4	3 5 6
5	3	5	1	5	5	5	3	5	3 4 7
6	1	6	1	6	6	6	1	6	4
7	1	7	1	7	7	7	1	7	5

VCOLOR	LIST	IDXLIST
0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
[0 1 2 **0** 4 5 6 7]	[0 1 2 **7** 4 5 6 **3**]	[0 1 2 **7** 4 5 6 **3**]

Since we are dealing with simple graphs, any edge between two vertices belonging to the same color set is deleted and no more than one edge is allowed between a color set and another. Clearly, such an operation can be implicitly taken back, but is also more time and space consuming than simple vertex deletion.

This technique makes it possible to implement the vertex folding operation, introduced in [1], for the parameterized VC algorithm.

4.4 The Vertex Cover Folding Operation

Let (G, k) be an instance of the Vertex Cover problem and let $u \in V(G)$ be a degree-two vertex with neighbors v and w. If v and w are adjacent, then there is a minimum vertex cover that contains v and w (and not u). So it is safe to delete u, add v and w to the potential solution and decrement k by two. In the case where v and w are non-adjacent, an equivalent Vertex Cover instance is obtained by contracting edges uv and uw and decrementing k by one. This latter operation is known as degree-two vertex folding.

As we shall see, applying the coloring technique to implement vertex folding considerably improves the runtime on certain recalcitrant instances, but slows down the computation on graphs where folding rarely occurs. In such cases, the overhead of maintaining color-sets is a drawback. Note that folding alone made it possible to obtain a worst-case run time of $O^*(1.285^k)$ in [1]. Yet, our results show that excluding folding from the same algorithm is faster on a large number of instances, especially real ones.

5 Experimental Results

Four different versions were implemented for the VERTEX COVER algorithm. AL_VC_OPT and HYBRID_VC_OPT are two generic search-tree optimization versions using the adjacency-list and hybrid graph representations respectively. In Table 1, the running times for both versions are reported for a number of DIMACS graphs. HYBRID_VC_PARM is a parameterized hybrid version that does

Table 1. AL_VC_OPT vs. HYBRID_VC_OPT (no folding)

Graph	$\|V\|$	$\|E\|$	$\|C\|$	AL_VC_OPT	HYBRID_VC_OPT
brock800_1.clq	800	207505	790	1 min 54 sec	32 sec
p_hat300-1.clq	300	10933	261	1 min 25 sec	41 sec
p_hat500-1.clq	500	31569	450	3 hr 48 min	1 hr 23 min
p_hat500-2.clq	500	62946	464	40 sec	12 sec
p_hat700-1.clq	700	60999	635	$>$ 1 week	93 hr 20 min
p_hat700-2.clq	700	121728	651	15 min 10 sec	3 min 44 sec
p_hat700-3.clq	700	183010	690	20 sec	6 sec
p_hat1000-2.clq	1000	244799	946	31 hr 26 min	5 hr 28 min
p_hat1000-3.clq	1000	371746	989	2 min 47 sec	48 sec
p_hat1500-3.clq	1500	847244	1488	20 min 57 sec	5 min 3 sec

Table 2. HYBRID_VC_PARM vs. HYBRID_VCF_PARM (with folding) on a 4-regular graph having 300 vertices and 600 edges

Vertex Cover Size ($\|K\|$)	Answer	No Folding	With Folding
192	yes	14 sec	$<$ 1 sec
191	yes	17 sec	6 sec
190	yes	2 hr 14 min	6 min 27 sec
165	no	$>$ 4 days	46 min 56 sec
160	no	38 hr 2 min	2 min 32 sec

Table 3. AL_DS_OPT vs. HYBRID_DS_OPT

Graph	$\|V\|$	$\|E\|$	$\|D\|$	AL_DS_OPT	HYBRID_DS_OPT
rgraph1	100	400	16	21 min 4 sec	4 min 11 sec
rgraph2	100	600	11	5 min 5 sec	53 sec
rgraph3	100	1500	6	41 sec	5 sec
rgraph4	150	1200	14	16 hr 46 min	2 hr 27 min
rgraph5	150	1500	11	3 hr 31 min	28 min 20 sec
rgraph6	150	3000	6	2 min 8 sec	12 sec
rgraph7	150	3000	7	27 min 16 sec	2 min 1 sec
rgraph8	200	4500	9	5 hr 44 min	30 min 8 sec
rgraph9	200	5000	8	1 hr 20 min	6 min 46 sec
rgraph10	200	6000	6	1 hr 36 min	7 min 13 sec
rgraph11	200	12000	4	6 min 49 sec	17 sec
rgraph12	250	9000	8	14 hr 37 min	56 min 53 sec
rgraph13	250	10000	7	1 hr 30 min	5 min 34 sec
rgraph14	250	12000	5	4 hr 41 min	16 min 19 sec
rgraph15	250	24000	3	19 sec	$<$ 1 sec
rgraph16	300	22461	4	8 min 31 sec	17 sec
rgraph17	300	22258	4	28 min 32 sec	1 min 10 sec
rgraph18	300	11063	8	133 hr 38 min	5 hr 54 min
rgraph19	300	11287	8	$>$ 7 days	8 hr 14 min
rgraph20	1000	374633	3	4 min 37 sec	2 min 29 sec
rgraph21	1000	374552	3	28 min 37 sec	6 min 36 sec

not take advantage of vertex folding, while HYBRID_VCF_PARM is a parameterized version implemented using the coloring technique described in the previous section for folding.

In general, the folding technique is at most two times slower than the simple generic branching algorithm. It gets faster as the difference between the highest and lowest vertex-degrees gets smaller. In particular, applying vertex folding, via our coloring technique, is much faster on regular graphs. To illustrate, tests were run on a 4-regular graph, by varying the input parameter, and results are reported in Table 2.

As for the DOMINATING SET problem, AL_DS_OPT denotes the optimization version using the adjacency-lists representation and HYBRID_DS_OPT the

Table 4. AL_DS_OPT vs. HYBRID_DS_OPT

Graph	$\|V\|$	$\|E\|$	$\|D\|$	AL_DS_OPT	HYBRID_DS_OPT
GDS3211.96	4636	11249	1567	3 min 57 sec	1 min 3 sec
GDS3221.95	5759	43991	1551	11 min 55 sec	3 min 6 sec
GDS3221.94	8517	131498	2042	1 hr 5 min	11 min 8 sec
GDS3221.93	11065	315488	2427	3 hr 58 min	26 min 27 sec
GDS3221.92	13712	649073	2758	$>$ 6 hours	54 min 43 sec

optimization version using the hybrid graph representation. Running times of the DS implementations on random graphs, with various densities, are given in Table 3. In addition, real DS instances for biological problems were obtained from the Gene Expression Omnibus (GEO) data-sets available at `http://www.ncbi.nlm.nih.gov` and the results are shown in Table 4. The raw data (SOFT) files were transformed into simple unweighted graphs using Pearson's coefficients and appropriate thresholding. The threshold value used for each graph appears in the file extension in Table 4.

All codes were implemented in standard C, and experiments were run on two types of machines (in two labs): Intel Core2 Duo 2327 MHz and Intel Xeon Processor X5550 (Nahalem) 2.66 GHz Quad Core. However, the numbers reported in each row were obtained on the same architecture.

6 Conclusion

We presented a hybrid graph representation that efficiently trades space for time and facilitates many common graph operations required during recursive backtracking. Experiments on both Vertex Cover and Dominating Set showed the utility of using this dynamic data structure. The running times of the same algorithm were shown to be consistently reduced, sometimes from days to hours.

The main focus in this paper was on operations reducing the original graph size, such as vertex deletion and edge contraction. However, some algorithms require operations that do not decrease the size of an input graph. Such operations are harder to implement. Edge addition is a notable example that remains to be considered.

References

1. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: Further observations and further improvements. Journal of Algorithms 41, 313–324 (2001)
2. Chen, J., Liu, L., Jia, W.: Improvement on vertex cover for low-degree graphs. Networks 35(4), 253–259 (2000)
3. Engebretsen, L., Holmerin, J.: Clique is hard to approximate within $n^{1-o(1)}$. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 2–12. Springer, Heidelberg (2000)

4. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: domination - a case study. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 191–203. Springer, Heidelberg (2005)
5. Fomin, F.V., Grandoni, F., Kratsch, D.: Solving connected dominating set faster than 2^n. Algorithmica 52(2), 153–166 (2008)
6. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm (SODA), New York, USA, pp. 18–25 (2006)
7. Fomin, F.V., Kratsch, D., Woeginger, L., Woeginger, G.J.: Exact (exponential) algorithms for the dominating set problem. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 245–256. Springer, Heidelberg (2004)
8. Grandoni, F.: A note on the complexity of minimum dominating set. J. Discrete Algorithms 4(2), 209–214 (2006)
9. Hastad, J.: Clique is hard to approximate within $n^{(1-\epsilon)}$. Acta Mathematica, 627–636 (1996)
10. Randerath, B., Schiermeyer, I.: Exact algorithms for minimum dominating set. Technical report, Zentrum für Angewandte Informatik Köln, Lehrstuhl Speckenmeyer (2004)
11. van Rooij, J.M., Bodlaender, H.L.: Exact algorithms for edge domination. Technical Report UU-CS-2007-051, Department of Information and Computing Sciences, Utrecht University (2007)
12. van Rooij, J.M., Nederlof, J., van Dijk, T.C.: Inclusion/exclusion meets measure and conquer: Exact algorithms for counting dominating sets. Technical Report UU-CS-2008-043, Department of Information and Computing Sciences, Utrecht University (2008)
13. van Rooij, J.M., Bodlaender, H.L.: Design by measure and conquer, a faster exact algorithm for dominating set. In: Albers, S., Weil, P. (eds.) STACS. LIPIcs, vol. 1, pp. 657–668. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2008)

On Tractable Exponential Sums

Jin-Yi Cai[1,⋆], Xi Chen[2,⋆⋆], Richard Lipton[3], and Pinyan Lu[4]

[1] University of Wisconsin-Madison
jyc@cs.wisc.edu
[2] University of Southern California
csxichen@gmail.com
[3] Georgia Institute of Technology
richard.lipton@cc.gatech.edu
[4] Microsoft Research Asia
pinyanl@microsoft.com

Abstract. We consider the problem of evaluating certain exponential sums. These sums take the form

$$\sum_{x_1,x_2,\ldots,x_n \in \mathbb{Z}_N} e^{\frac{2\pi i}{N} f(x_1,x_2,\ldots,x_n)},$$

where each x_i is summed over a ring $\mathbb{Z}_N$, and $f(x_1, x_2, \ldots, x_n)$ is a multivariate polynomial with integer coefficients. We show that the sum can be evaluated in polynomial time in n and $\log N$ when f is a quadratic polynomial. This is true even when the factorization of N is unknown. Previously, this was known for a prime modulus N. On the other hand, for very specific families of polynomials of degree ≥ 3 we show the problem is #P-hard, even for any fixed prime or prime power modulus. This leads to a complexity dichotomy theorem — a complete classification of each problem to be either computable in polynomial time or #P-hard — for a class of exponential sums. These sums arise in the classifications of graph homomorphisms and some other counting CSP type problems, and these results lead to complexity dichotomy theorems. For the polynomial-time algorithm, Gauss sums form the basic building blocks; for the hardness result we prove group-theoretic necessary conditions for tractability.

1 Introduction

Exponential sums are among the most studied objects in Number Theory [1,2,3]. They have fascinating properties and innumerable applications. Recently they have also played a pivotal role in the study of computational complexity of graph homomorphisms [4,5].

⋆ Supported by NSF CCF-0830488 and CCF-0914969.

⋆⋆ Most of the work done while the author was a postdoc at the Institute for Advanced Study and Princeton University. Supported by Grants CCF-0832797, DMS-0635607, and the USC Viterbi School of Engineering Startup Fund (from Shang-Hua Teng).

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 148–159, 2010.

The most fundamental and well-known among exponential sums are those named after Gauss. Let p be an odd prime, and $\omega_p = e^{2\pi i/p}$ be the p-th primitive root of unity. Then the Gauss sum over $\mathbb{Z}_p$ is

$$G = \sum_{t \in \mathbb{Z}_p} \left(\frac{t}{p}\right) \omega_p^t, \quad \text{where } \left(\frac{t}{p}\right) \text{ is the Legendre symbol.} \tag{1}$$

In this paper, we will need to use a more general form of the Gauss sum which will be defined later in Section 1. Another well-known expression for G in (1) is

$$G = \sum_{x \in \mathbb{Z}_p} \left(\omega_p\right)^{x^2}.$$

Gauss also knew the remarkable equality $G^2 = (-1)^{(p-1)/2} p$; i.e.,

$$G = \pm\sqrt{p} \quad \text{if } p \equiv 1 \pmod 4, \quad \text{and} \quad G = \pm i\sqrt{p} \quad \text{if } p \equiv 3 \pmod 4. \tag{2}$$

In particular, we have $|G| = \sqrt{p}$, which is an expression that the p terms in the sum G are somewhat "randomly" distributed on the unit circle (but note that the equality is exact). However, the truly amazing fact is that, in all cases, the plus sign (+) always holds in (2). Gauss recorded this conjecture in his diary in May 1801, and on August 30, 1805 Gauss recorded that a proof of the "very elegant theorem mentioned in 1801" had finally been achieved.

In this paper we consider the computational complexity of evaluating exponential sums of the form

$$Z(N, f) = \sum_{x_1, x_2, \ldots, x_n \in \mathbb{Z}_N} e^{\frac{2\pi i}{N} f(x_1, x_2, \ldots, x_n)},$$

where each x_i is summed over a ring $\mathbb{Z}_N$ and $f(x_1, x_2, \ldots, x_n)$ is a multivariate polynomial with integer coefficients. The output of the computation is an algebraic number, in the cyclotomic field $\mathbb{Q}(e^{2\pi i/N})$. Any canonical representation of the output algebraic number will be acceptable [6,7]. These sums are natural generalizations of the sums considered by Gauss and with arbitrary polynomials f, they have also played important roles in the development of number theory.

Our main results are as follows: We show that the sum $Z(N, f)$ can be evaluated in polynomial time when f is a quadratic polynomial. The computational complexity is measured in terms of n, $\log N$, and the number of bits needed to describe f. While it is known that $Z(N, f)$ can be computed efficiently when N is a prime [8], our algorithm works for any composite modulus N, even without knowing its prime factorization. On the other hand, for very specific families of polynomials of degree ≥ 3, we show the problem is #P-hard even for any fixed prime or prime power modulus. This leads to a complexity dichotomy theorem — a complete classification of each problem to be either computable in polynomial time or #P-hard — for a class of exponential sums.

For the polynomial-time algorithm, we employ an iterative process to eliminate one variable at a time. Gauss sums form the basic building blocks. The fact that we know the exact answer to the Gauss sum, *including the sign*, is crucial.

It turns out that the situation is different for an odd or an even modulus N. A natural idea is to deal with each prime power in the modulus N separately, and combine the answers by Chinese remaindering. It turns out that the algorithm is more difficult for a modulus which is a power of 2, than for an odd prime power. A more fundamental difficulty arises when N is large and its prime factorization is unknown. We overcome this difficulty as follows: (1) Factor out all powers of 2 in N and deal with it separately. (2) Operate in the remaining odd modulus *as if it were* an odd prime power; whenever this operational commingling encounters an obstacle, we manage to discover a non-trivial factorization of the modulus N into relatively prime parts. In that case we recurse.

Theorem 1. *Let N be any positive integer and $f \in \mathbb{Z}[x_1, \ldots, x_n]$ be a quadratic polynomial in n variables $x_1, \ldots, x_n$. Then the sum $Z(N, f)$ can be evaluated in polynomial time in n, $\log N$, and the number of bits needed to describe f.*

Previously, it was known that for quadratic polynomials f, the sum can be computed in polynomial time, if N is a prime [8]. An algorithm with running time $O(n^3)$ can also be found in the paper by Ehrenfeucht and Karpinski [9]. Compared to these algorithms, ours works for any N even if it is given as a part of the input and its factorization is unknown. It was also suggested that there is a reduction from root counting. One can express the sum as

$$\sum_{k=0}^{N-1} \#[f = k] \cdot e^{2\pi i k/N}.$$

If N is polynomially bounded and if one can compute $\#[f = k]$ for all k, then one can compute the sum. But this works only when N is small. Our results are for general N (polynomial time in the length $\log N$). In our algorithm, Gauss sums play a crucial role. Any claim to the contrary amounts to an independent proof of Gauss's sign formula (that "very elegant theorem mentioned in 1801"), since it is not only a crucial building block of our algorithm, but also a special case of the algorithm. We also note that our treatment for the case when N is a power of 2 is significantly different than previous work. No simple adaptation of ideas from Sylvester's law of inertia seems to work.

For the hardness part, we give several successively more stringent necessary conditions for a class of polynomials to be tractable. The first necessary condition involves the rank of an associated matrix, and the proof uses the widely applicable dichotomy theorem of Bulatov and Grohe [10] on counting graph homomorphisms over non-negative weighted graphs. The second condition involves linear independence and orthogonality. The third and much more stringent necessary condition is group-theoretic in nature; it asserts that the set of row vectors of a certain complex matrix must form a group. In the paper [4], Goldberg et al. had proved a similar condition for $\{-1, +1\}$-matrices, in the study of graph homomorphisms over real weighted graphs. Finally, in subsection 4.1, we give a Generalized Group Condition which leads to a complexity dichotomy.

Previously, it was shown by Ehrenfeucht and Karpinski [9] that for any fixed prime N, the problem of computing $Z(N, f)$ for general cubic polynomials f is #P-hard [9]. However, our tests in Section 4 are more powerful. They allow us

to prove the #P-hardness of $Z(N, f)$ even if f belongs to some very restricted families of polynomials, since they fail one of the tests in Section 4.

These sums arise recently in the classifications of graph homomorphisms as well as some other counting CSP type problems (include both CSP and Holant Problems). For example, the special case when $N = 2$ is a key component of the dichotomy of Goldberg et al. [4] for graph homomorphisms over real weighted graphs. It implies that the partition function $Z_{\mathbf{H}}(\cdot)$ (see the definition in section 4) with $H_{1,1} = H_{1,2} = H_{2,1} = 1$ and $H_{2,2} = -1$, which has been an obstacle to the dichotomy theorem of Bulatov and Grohe [10] and was left open for some time, can actually be computed in polynomial time.

Preliminaries

Let $\omega_N = e^{2\pi i/N}$ denote the N-th primitive root of unity. Let $N = N_1 \cdot N_2$ be a non-trivial factorization, namely $N_1, N_2 > 1$. Suppose N_1 and N_2 are relatively prime, then there exist integers a and b such that $bN_1 + aN_2 = 1$. It follows that

$$Z(N, f) = Z(N_1, af) \cdot Z(N_2, bf). \tag{3}$$

Therefore, if we know a non-trivial factorization of N into relatively prime factors N_1 and N_2, then the problem $Z(N, f)$ decomposes. In particular, we can factor $N = 2^k N'$, where N' is odd. Thus we can treat the problems $Z(2^k, \cdot)$ and $Z(N', \cdot)$ separately. In Section 2, we give an algorithm for the case when N is odd and in Section 3 we deal with the case when $N = 2^k$.

Our algorithm crucially relies on the fact that the following general form of Gauss sum $G(a,b)$ can be computed in polynomial time in $\log a$ and $\log b$, even without knowing their prime factorizations. Let a, b be non-zero integers with $b > 0$ and $\gcd(a, b) = 1$. Then $G(a, b)$ denotes the following sum:

$$G(a, b) = \sum_{x \in \mathbb{Z}_b} \omega_b^{ax^2}.$$

The algorithm for computing $G(a, b)$ can be found in the full version [11].

2 Odd Modulus

First, we present a polynomial-time algorithm for the case when N is odd. Let

$$f(x_1, \ldots, x_n) = \sum_{i \le j \in [n]} c_{i,j} x_i x_j + \sum_{i \in [n]} c_i x_i + c_0. \tag{4}$$

We may assume $c_0 = 0$ because it only contributes a constant factor to $Z(N, f)$. For each non-zero coefficient $c = c_{i,j}$ or c_i of f, we compute the greatest common divisor $g = \gcd(N, c^{\lfloor \log_2 N \rfloor})$. Note that if $\mathrm{ord}_p N$ is the exact order of a prime p in N, then $N \ge p^{\mathrm{ord}_p N}$ and thus $\mathrm{ord}_p N \le \lfloor \log_2 N \rfloor$. Hence if c shares any prime p with N, but not all the prime factors of N, then g has the factor $p^{\mathrm{ord}_p N}$, and $N = g \cdot (N/g)$ is a non-trivial factorization of N into two relatively prime factors. We can test for each non-zero $c = c_{i,j}$ or c_i whether $N = g \cdot (N/g)$ gives us a non-trivial factorization of N into two relatively prime factors.

By (3), if for some c, we did find such a factorization $N = N_1 \cdot N_2$, then the problem decomposes into two subproblems $Z(N_1, \cdot)$ and $Z(N_2, \cdot)$. There can be at most a linear number $\log_2 N$ many such subproblems, and a polynomial-time algorithm for each subproblem will give a polynomial-time algorithm for $Z(N, \cdot)$. Therefore, in the following we assume for each non-zero coefficient $c = c_{i,j}$ or c_i, either $\gcd(N, c) = 1$ or c has all prime factors of N, and we know, by computing the gcd, which case it is for each coefficient c. We consider the following cases.

Case 1. There exists some diagonal coefficient $c_{i,i}$ relatively prime to N.

Without loss of generality we assume $c_{1,1}$ is relatively prime to N. Then $c_{1,1}$ is invertible in $\mathbb{Z}_N$. Since N is odd, 2 is also invertible. Denote by $c'_{1,i}$ an integer such that $c'_{1,i} \equiv (2c_{1,1})^{-1} c_{1,i} \pmod N$, for $2 \le i \le n$. We have, modulo N,

$$f(x_1, \ldots, x_n) = c_{1,1}\left[x_1^2 + 2x_1(c'_{1,2}x_2 + \ldots + c'_{1,n}x_n)\right] + \textstyle\sum_{2 \le i \le j \le n} c_{i,j}x_ix_j + \sum_{i \in [n]} c_i x_i.$$

Let $g(x_2, \ldots, x_n) = c'_{1,2}x_2 + \ldots + c'_{1,n}x_n$. Then we can write f as

$$f = c_{1,1}(x_1 + g)^2 + c_1(x_1 + g) + h,$$

where h is some quadratic polynomial in $x_2, \ldots, x_n$. If we substitute $y = x_1 + g$ for x_1, then for any fixed $x_2, \ldots, x_n \in \mathbb{Z}_N$, when x_1 takes all the values in $\mathbb{Z}_N$, y also takes all the values in $\mathbb{Z}_N$. Hence, we have

$$Z(N, f) = \sum_{x_2, \ldots, x_n \in \mathbb{Z}_N} \sum_{y \in \mathbb{Z}_N} \omega_N^{c_{1,1}y^2 + c_1 y + h(x_2, \ldots, x_n)}.$$

Completing the square again, $c_{1,1}y^2 + c_1 y = c_{1,1}(y + (2c_{1,1})^{-1}c_1)^2 + c'$, where

$$c' = -c_1^2/(4c_{1,1}) \in \mathbb{Z}_N \quad \text{and} \quad Z(N, f) = \sum_{x_2, \ldots, x_n \in \mathbb{Z}_N} \sum_{z \in \mathbb{Z}_N} \omega_N^{c_{1,1}z^2 + h'(x_2, \ldots, x_n)},$$

where $h'(x_2, \ldots, x_n) = h(x_2, \ldots, x_n) + c'$ is an explicitly computed quadratic polynomial in $x_2, \ldots, x_n$. It then follows that $Z(N, f) = Z(N, h') \cdot G(c_{1,1}, N)$, where h' has (at least) one fewer variable than f and the Gauss sum $G(c_{1,1}, N)$ can be computed in polynomial time. This completes the proof of Case 1.

Case 2. No $c_{i,i}$ is relatively prime to N but there exist some $i, j : 1 \le i < j \le n$ such that $\gcd(c_{i,j}, N) = 1$. By our earlier assumption, for every prime factor p of N, p divides every $c_{i,i}$ for all $1 \le i \le n$.

The existence of $c_{i,j}$ for some $i, j : 1 \le i < j \le n$ implies that in particular $n \ge 2$. Without loss of generality, we assume $\gcd(c_{1,2}, N) = 1$. Now we perform the following substitution: $x_1 = y_1 + y_2$, $x_2 = y_1 - y_2$, and x_i are unchanged for any $2 < i \le n$ if $n > 2$. This transformation is a 1-1 correspondence from $\mathbb{Z}_N^n$ to itself with inverse $y_1 = (x_1 + x_2)/2$ and $y_2 = (x_1 - x_2)/2$ because 2 is invertible in $\mathbb{Z}_N$. Since the transformation is linear it does not change the degree of f. It is easily checked that the coefficient of y_1^2 in the new polynomial is $c_{1,1} + c_{2,2} + c_{1,2}$. Since $c_{1,1}$ and $c_{2,2}$ have all the prime factors of N, $c_{1,1} + c_{2,2} + c_{1,2}$ is relatively prime to N. This transformation reduces the computation of $Z(N, f)$ to Case 1.

Case 3. No coefficients $c_{i,j}$ of f, where $1 \le i \le j \le n$, are relatively prime to N. However, there exists a c_i relatively prime to N, for some $i : 1 \le i \le n$. Without loss of generality, assume $\gcd(c_1, N) = 1$. Let p be a prime divisor of N, then

$$p \mid c_{1,1}, \ldots, c_{1,n} \quad \text{and yet} \quad p \nmid c_1. \tag{5}$$

Let $k = \mathrm{ord}_p N$ be the exact order of p in N with $k \ge 1$. Write $N = p^k N_1$, then $\gcd(p, N_1) = 1$, and for some integers a and b, we have $bp^k + aN_1 = 1$. By (3), $Z(N, f) = Z(p^k, af) \cdot Z(N_1, bf)$. Note that $\gcd(a, p) = 1$. Hence the condition (5) for the coefficients of f also holds for af. We will show $Z(p^k, af) = 0$. For notational simplicity, we will write below f for af.

$$Z(p^k, f) = \sum_{x_2, \ldots, x_n \in \mathbb{Z}_{p^k}} \omega_{p^k}^{\sum_{2 \le i \le j \le n} c_{i,j} x_i x_j + \sum_{2 \le i \le n} c_i x_i} \sum_{x_1 \in \mathbb{Z}_{p^k}} \omega_{p^k}^{\sum_{1 \le i \le n} c_{1,i} x_1 x_i + c_1 x_1}.$$

We fix any $x_2, \ldots, x_n \in \mathbb{Z}_{p^k}$, and consider the inner sum over x_1. If $k = 1$, then all terms $c_{1,i} x_1 x_i$ disappear, and because $p \nmid c_1$, the inner sum is equal to 0.

Now suppose $k > 1$. We repeat the sum for p times with $x^{(j)} = x_1 + j \cdot p^{k-1}$ where $0 \le j \le p - 1$. Then by (5), we have $c_{1,i} x_1 x_i \equiv c_{1,i} x^{(j)} x_i \pmod{p^k}$ and

$$\sum_{x_1 \in \mathbb{Z}_{p^k}} \omega_{p^k}^{\sum_{1 \le i \le n} c_{1,i} x_1 x_i + c_1 x_1} = \frac{1}{p} \sum_{x_1 \in \mathbb{Z}_{p^k}} \omega_{p^k}^{\sum_{1 \le i \le n} c_{1,i} x_1 x_i + c_1 x_1} \left(\sum_{j=0}^{p-1} \omega_p^{j c_1} \right).$$

By $p \nmid c_1$, the geometric sum $\sum_{j=0}^{p-1} \omega_p^{j c_1} = 0$. This finishes Case 3.

Case 4. No coefficients $c_{i,j}$ and c_ℓ of f, where $1 \le i \le j \le n$ and $1 \le \ell \le n$, are relatively prime to N.
By our earlier assumption, this means that every prime factor of N divides every coefficient $c_{i,j}$ and c_ℓ. Then we can find the joint gcd d of N with all these coefficients, which must at least contain every prime factor of N, and divide out d in the exponent. By $\omega_N^d = \omega_{N/d}$, we get $Z(N, f) = d \cdot Z(N/d, f')$ where $f' = f/d$ is the quadratic polynomial obtained from f by dividing every coefficient with d.This reduces the modulus from N to N/d.

By combining all the four cases, we get a polynomial-time algorithm for the case when N is odd.

3 Modulus Is a Power of 2

Next, we deal with the more difficult case when the modulus, denoted by q here, is a power of 2: $q = 2^k$ for some $k \ge 1$. We note that the property of an element $c \in \mathbb{Z}_q$ being even or odd is well-defined.

For the case when $k = 1$, $Z(q, f)$ is computable in polynomial time by [8]. So we always assume $k > 1$ below. The algorithm goes as follows. For each round, we show how to, in polynomial time, either

1. output the correct value of $Z(q, f)$; or

2. construct a new quadratic polynomial $g \in \mathbb{Z}_{q/2}[x_1, \ldots, x_n]$ and reduce the computation of $Z(q, f)$ to the computation of $Z(q/2, g)$; or
3. construct a new quadratic polynomial $g \in \mathbb{Z}_q[x_1, \ldots, x_{n-1}]$, and reduce the computation of $Z(q, f)$ to the computation of $Z(q, g)$.

This gives us a polynomial-time algorithm for evaluating $Z(q, f)$ since we know how to solve the two base cases when either $k = 1$ or $n = 0$ efficiently.

Suppose we have a polynomial $f \in \mathbb{Z}_q[x_1, \ldots, x_n]$ as in (4). Our first step is to transform f so that all the coefficients of its cross terms ($c_{i,j}$, where $1 \le i < j \le n$) and linear terms (c_i) are even. Assume f does not yet have this property. We let t be the smallest index in $[n]$ such that one of $\{c_t, c_{t,j} : j > t\}$ is odd. By separating out the terms involving x_t, we rewrite f as follows

$$f \;=\; c_{t,t} \cdot x_t^2 + x_t \cdot f_1(x_1, \ldots, \widehat{x_t}, \ldots, x_n) + f_2(x_1, \ldots, \widehat{x_t}, \ldots, x_n), \tag{6}$$

where f_1 is an affine linear function and f_2 is a quadratic polynomial. Both f_1 and f_2 here are over variables $\{x_1, \ldots, x_n\} - \{x_t\}$. Here the notation $\widehat{x_t}$ means that x_t does not appear in the polynomial. Moreover

$$f_1(x_1, \ldots, \widehat{x_t}, \ldots, x_n) \;=\; \textstyle\sum_{i<t} c_{i,t} x_i + \sum_{j>t} c_{t,j} x_j + c_t. \tag{7}$$

By the minimality of t, $c_{i,t}$ is even for all $i < t$ and at least one of $\{c_t, c_{t,j} : j > t\}$ is odd. We claim that

$$Z(q, f) \;=\; \sum_{x_1, \ldots, x_n \in \mathbb{Z}_q} \omega_q^{f(x_1, \ldots, x_n)} \;=\; \sum_{\substack{x_1, \ldots, x_n \in \mathbb{Z}_q \\ f_1(x_1, \ldots, \widehat{x_t}, \ldots, x_n) \equiv 0 \bmod 2}} \omega_q^{f(x_1, \ldots, x_n)}. \tag{8}$$

This is because

$$\sum_{\substack{x_1, \ldots, x_n \in \mathbb{Z}_q \\ f_1 \equiv 1 \bmod 2}} \omega_q^{f(x_1, \ldots, x_n)} \;=\; \sum_{\substack{x_1, \ldots, \widehat{x_t}, \ldots, x_n \in \mathbb{Z}_q \\ f_1 \equiv 1 \bmod 2}} \; \sum_{x_t \in \mathbb{Z}_q} \omega_q^{c_{t,t} x_t^2 + x_t f_1 + f_2}.$$

However, for any fixed $x_1, \ldots, \widehat{x_t}, \ldots, x_n$, the inner sum is equal to $\omega_q^{f_2}$ times

$$\sum_{x_t \in [0 : 2^{k-1} - 1]} \omega_q^{c_{t,t} x_t^2 + x_t f_1} + \omega_q^{c_{t,t}(x_t + 2^{k-1})^2 + (x_t + 2^{k-1}) f_1} = \left(1 + (-1)^{f_1}\right) \sum_{x_t} \omega_q^{c_{t,t} x_t^2 + x_t f_1} = 0,$$

since $f_1 \equiv 1 \bmod 2$. Note that we used $(x + 2^{k-1})^2 \equiv x^2 \pmod{2^k}$ when $k > 1$ in the first equation.

Recall that f_1 (see (7)) is an affine linear form of $\{x_1, \ldots, \widehat{x_t}, \ldots, x_n\}$. Also note that $c_{i,t}$ is even for all $i < t$ and one of $\{c_t, c_{t,j} : j > t\}$ is odd. We consider the following two cases.

In the first case, $c_{t,j}$ is even for all $j > t$ and c_t is odd, then f_1 is odd for any assignment $(x_1, \ldots, \widehat{x_t}, \ldots, x_n)$ in $\mathbb{Z}_q^{n-1}$. As a result, $Z(q, f) = 0$ by (8).

In the second case, there exists at least one $j > t$ such that $c_{t,j}$ is odd. Let $\ell > t$ be the smallest of such j's. Then we substitute the variable x_ℓ in f with a new variable x'_ℓ, where (as $c_{t,\ell}$ is odd, $c_{t,\ell}$ is invertible in $\mathbb{Z}_q$)

$$x_\ell \;=\; c_{t,\ell}^{-1} \left(2x'_\ell - \left(\textstyle\sum_{i<t} c_{i,t} x_i + \sum_{j>t, j \neq \ell} c_{t,j} x_j + c_t\right)\right). \tag{9}$$

and let f' denote the new quadratic polynomial in $\mathbb{Z}_q[x_1, \ldots, x_{\ell-1}, x'_\ell, x_{\ell+1}, \ldots, x_n]$. We claim that

$$Z(q, f') = 2 \cdot Z(q, f) = 2 \cdot \sum_{\substack{x_1, \ldots, x_n \in \mathbb{Z}_q \\ f_1 \equiv 0 \bmod 2}} \omega_q^{f(x_1, \ldots, x_n)}.$$

To this end, we define the following map from $\mathbb{Z}_q^n$ to $\mathbb{Z}_q^n$: $(x_1, \ldots, x'_\ell, \ldots, x_n) \mapsto (x_1, \ldots, x_\ell, \ldots, x_n)$, where x_ℓ satisfies (9). It is easy to check that the range of this map is exactly the set of $(x_1, \ldots, x_\ell, \ldots, x_n)$ in $\mathbb{Z}_q^n$ such that f_1 is even. Moreover, for every such tuple $(x_1, \ldots, x_\ell, \ldots, x_n)$ the number of its preimages in $\mathbb{Z}_q^n$ is exactly 2. The claim then follows.

As a result, to compute $Z(q, f)$, we only need to compute $Z(q, f')$, and the advantage of the new polynomial f' over f is the following property. The proof of Property 1 can be found in the full version [11].

Property 1. For every cross and linear term that involves $x_1, \ldots, x_t$, its coefficient in f' is even.

To summarize, after substituting x_ℓ with x'_ℓ using (9), we obtain a quadratic polynomial f' such that $Z(q, f') = 2 \cdot Z(q, f)$ and for all cross and linear terms that involve $x_1, \ldots, x_t$, its coefficient in f' is even. We can repeat this substitution procedure on f': either we show that $Z(q, f')$ is trivially 0 or we get a quadratic polynomial f'' such that $Z(q, f'') = 2 \cdot Z(q, f')$ and the parameter t increases by at least one. As a result, given any quadratic polynomial f, we can, in polynomial time, either show that $Z(q, f)$ is 0 or construct a new quadratic polynomial $g \in \mathbb{Z}_q[x_1, \ldots, x_n]$ such that $Z(q, f) = 2^d \cdot Z(q, g)$ for some known integer $d \leq n$, and every cross term and linear term of g has an even coefficient.

For notational simplicity, we can just assume that the given f in (4) already satisfies this condition. (Or equivalently, we rewrite f for g.) We will show that, given such a polynomial f in n variables, we can reduce it either to the computation of $Z(q/2, f')$, in which f' is a quadratic polynomial in n variables; or to the computation of $Z(q, f'')$, in which f'' is a quadratic polynomial in $n-1$ variables. We consider the following two cases: $c_{i,i}$ is even for all $i \in [n]$; or at least one of the $c_{i,i}$'s is odd.

In the first case, we know $c_{i,j}$ and c_i are even for all $1 \leq i \leq j \leq n$. We use $c'_{i,j}$ and c'_i to denote integers in $[0 : 2^{k-1} - 1]$ such that $c_{i,j} \equiv 2c'_{i,j} \pmod q$ and $c_i \equiv 2c'_i \pmod q$, respectively. Then,

$$Z(q, f) = \omega_q^{c_0} \cdot \sum_{x_1, \ldots, x_n \in \mathbb{Z}_q} \omega_q^{2\left(\sum_{i \leq j \in [n]} c'_{i,j} x_i x_j + \sum_{i \in [n]} c'_i x_i\right)} = 2^n \cdot \omega_q^{c_0} \cdot Z(2^{k-1}, f'),$$

where

$$f' = \textstyle\sum_{i \leq j \in [n]} c'_{i,j} x_i x_j + \sum_{i \in [n]} c'_i x_i$$

is a quadratic polynomial over $\mathbb{Z}_{q/2} = \mathbb{Z}_{2^{k-1}}$. This reduces the computation of $Z(q, f)$ to $Z(q/2, f')$.

In the second case, without loss of generality, we assume $c_{1,1}$ is odd, then

$$f = c_{1,1}(x_1^2 + 2x_1 f_1) + f_2 = c_{1,1}(x_1 + f_1)^2 + f',$$

where f_1 is an affine linear form, and both f_2 and f' are quadratic polynomials, all of which are over $x_2, \ldots, x_n$. We are able to do this because $c_{1,j}$ and c_1, for all $j \geq 2$, are even. Now we have

$$Z(q,f) = \sum_{x_1,\ldots,x_n \in \mathbb{Z}_q} \omega_q^{c_{1,1}(x_1+f_1)^2+f'} = \sum_{x_2,\ldots,x_n \in \mathbb{Z}_q} \omega_q^{f'} \cdot \sum_{x_1 \in \mathbb{Z}_q} \omega_q^{c_{1,1}(x_1+f_1)^2} = G(c_{1,1},q) \cdot Z(q,f').$$

The last equation is because the sum over $x_1 \in \mathbb{Z}_q$ is independent of the value of f_1. This reduces the computation of $Z(q,f)$ to $Z(q,f')$, and f' is a quadratic polynomial in $n-1$ variables.

To sum up, given any quadratic f, we can, in polynomial time, either output the correct value of $Z(q,f)$; or reduce one of the two parameters, k or n, by at least 1. This gives us a polynomial-time algorithm for $Z(q,f)$ when $q = 2^k$.

4 #P-Hardness

We first introduce the definition of a *partition function* $Z_{\mathbf{A}}(\cdot)$ [12,13,14,10,15], where $\mathbf{A}$ is a symmetric complex matrix. We give four necessary conditions on the matrix $\mathbf{A}$ for the problem of computing $Z_{\mathbf{A}}(\cdot)$ being *not* #P-hard. Then we demonstrate the wide applicability of these four conditions by reducing $Z_{\mathbf{A}}(\cdot)$, for some appropriate $\mathbf{A}$, to $Z(N,f)$ and proving that even computing $Z(N,f)$ for some very restricted families of polynomials over a fixed modulus N is #P-hard. Finally, we show that, for a large class of problems defined using $Z(N,f)$, these conditions actually cover all the #P-hard cases. Together with the polynomial-time algorithm presented in Section 2 and 3, they imply an explicit complexity dichotomy theorem for this class.

Let $\mathbf{A} \in \mathbb{C}^{m \times m}$ be a symmetric $m \times m$ matrix, then we define the partition function $Z_{\mathbf{A}}(\cdot)$ as follows: Given any undirected graph $G = (V,E)$ (Here G is allowed to have multi-edges but no self loops)

$$Z_{\mathbf{A}}(G) = \sum_{\xi: V \to [m]} \mathrm{wt}_{\mathbf{A}}(G,\xi), \quad \text{where} \quad \mathrm{wt}_{\mathbf{A}}(G,\xi) = \prod_{(u,v) \in E} A_{\xi(u),\xi(v)}. \tag{10}$$

The complexity of $Z_{\mathbf{A}}(\cdot)$, for various $\mathbf{A}$, has been studied intensely [12,13,14,10,15]. We need the following lemma which can be proved following an important result of Bulatov and Grohe [10]. The proof uses the technique of Valiant [16,17] called interpolation, which is omitted here.

Lemma 1 (The Rank-1 Condition). *Let $\mathbf{A} \in \mathbb{C}^{m \times m}$ be a symmetric matrix and let $\mathbf{A}'$ be the matrix such that $A'_{i,j} = |A_{i,j}|$ for all i,j. If there exists a 2×2 sub-matrix $\mathbf{B}$ of $\mathbf{A}'$, such that, $\mathbf{B}$ is of full rank and at least three of the four entries of $\mathbf{B}$ are non-zero, then computing $Z_{\mathbf{A}}(\cdot)$ is* #P-hard.

We can use lemma 1 to prove a stronger necessary condition for $Z_{\mathbf{A}}(\cdot)$ being *not* #P-hard. The proof can be found in the full version [11]. In the statement below, we let $\mathbf{A}_{i,*}$ denote the i-th row vector of $\mathbf{A}$. We say a matrix $\mathbf{A}$ is *M-discrete*, for some integer $M \geq 1$, if every entry of $\mathbf{A}$ is an M-th root of unity.

Lemma 2 (Orthogonality). *Let M be a positive integer and let $\mathbf{A}$ be a symmetric and* M-discrete *$m \times m$ matrix. If there exist $i \neq j \in [m]$ such that $\mathbf{A}_{i,*}$ and $\mathbf{A}_{j,*}$ are* neither *linearly dependent* nor *orthogonal, then $Z_{\mathbf{A}}(\cdot)$ is* #P-hard.

Next we prove a much stronger *group-theoretic* necessary condition for $Z_{\mathbf{A}}(\cdot)$ being not #P-hard, where $\mathbf{A}$ is any *discrete unitary matrix* as defined below. A similar condition was first used by Goldberg et al. in [4] for $\{+1, -1\}$-matrices, in the study of $Z_{\mathbf{A}}(\cdot)$ over real matrices. In the rest of this section, we will use $[0 : m-1]$ to index the rows and columns of an $m \times m$ matrix for convenience.

Definition 1 (Discrete Unitary Matrix). *Let $\mathbf{A} \in \mathbb{C}^{m \times m}$ be an $m \times m$ symmetric complex matrix. We say $\mathbf{A}$ is an* M-discrete unitary matrix*, for some positive integer M, if it is* M-discrete *and satisfies*

— $\forall\, i \in [0 : m-1]$, $A_{1,i} = A_{i,1} = 1$; $\forall\, i \neq j$, $\langle \mathbf{A}_{i,*}, \mathbf{A}_{j,*} \rangle = 0$, *where*

$$\langle \mathbf{A}_{i,*}, \mathbf{A}_{j,*} \rangle = \sum_{k \in [m]} \mathbf{A}_{i,k} \overline{\mathbf{A}_{j,k}}.$$

Given two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{C}^m$ we let $\mathbf{x} \circ \mathbf{y}$ denote their Hadamard product $\mathbf{z}$: $\mathbf{z} = \mathbf{x} \circ \mathbf{y} \in \mathbb{C}^m$, where $z_i = x_i \cdot y_i$ for all i.

Lemma 3 (The Group Condition). *Let $\mathbf{A} \in \mathbb{C}^{m \times m}$ be an $m \times m$ symmetric M-discrete unitary matrix, for some positive integer M. Then computing $Z_{\mathbf{A}}(\cdot)$ is #P-hard, unless $\mathbf{A}$ satisfies the following* Group Condition*:*

— $\forall\, i, j \in [0 : m-1]$, $\exists\, k \in [0 : m-1]$ *such that* $\mathbf{A}_{k,*} = \mathbf{A}_{i,*} \circ \mathbf{A}_{j,*}$.

These three necessary conditions are very powerful and can be used to prove the #P-hardness of $Z(N, f)$, for some very restricted families of polynomials f over a fixed modulus N. We would like to say, e.g., evaluating $Z(N, f)$, when f contains terms $x_1x_2x_3$, is #P-hard. However, we have to be very careful; such complexity-theoretic statements are only meaningful for a sequence of polynomials, and not an individual polynomial. This motivates the following definition. Let $h \in \mathbb{Z}[x_1, \ldots, x_r]$ be a fixed polynomial (e.g., $h = x_1x_2x_3$, with $r = 3$). We say $f \in \mathbb{Z}[x_1, \ldots, x_n]$ is an *h-type* polynomial, if there exists an r-uniform hypergraph $G = (V, E)$ with $V = [n]$ such that (We allow G to have multi-edges, i.e., E is a multiset; and edges in E are *ordered* subsets of $[n]$ of cardinality r)

$$f(x_1, \ldots, x_n) = \sum_{(i_1, \ldots, i_r) \in E} h(x_{i_1}, \ldots, x_{i_r}). \tag{11}$$

Definition 2. *Let $q = p^t$ be a prime power and $h \in \mathbb{Z}[x_1, \ldots, x_r]$ be a polynomial. We use $\mathcal{S}[q, h]$ to denote the following problem: given an r-uniform hypergraph G, compute $Z(q, f)$, where f is the* h-type polynomial *defined by G.*

Using these three necessary conditions, it is easy to prove the #P-hardness of the following problems, with

$$h_1(x_1, x_2, x_3) = x_1x_2x_3, \quad h_2(x_1, x_2) = x_1^2x_2 \quad \text{and} \quad h_3(x_1, x_2) = x_1x_2 + x_1^2x_2^2.$$

Corollary 1. *For any fixed prime power $q = p^t$, $\mathcal{S}[q, h_1]$ is #P-hard; For any prime power $q \notin \{2, 4\}$, $\mathcal{S}[q, h_2]$ and $\mathcal{S}[q, h_3]$ are #P-hard.*

Proof. We will only prove the statement for $\mathcal{S}[q, h_3]$ here. For $\mathcal{S}[q, h_3]$, let $\mathbf{A}$ be the following $m \times m$ symmetric and q-discrete matrix:

$$A_{i,j} = \omega_q^{h_3(i,j)}, \qquad \text{for all } i, j \in [0 : q - 1]. \tag{12}$$

It is easy to see that $Z_{\mathbf{A}}(\cdot)$ is computationally equivalent to $\mathcal{S}[q, h_3]$. Moreover, when q is an odd prime power, the two vectors $\mathbf{A}_{0,*}$ and $\mathbf{A}_{1,*}$ are neither linearly dependent nor orthogonal and thus, by Lemma 2, $\mathcal{S}[q, h_3]$ is #P-hard. For the case when $q = 2^t$ and $t > 2$, it can be checked that $\mathbf{A}$ is q-discrete unitary but does not satisfy the Group Condition. Then by Lemma 3, $\mathcal{S}[q, h_3]$ is #P-hard.

4.1 A Dichotomy Theorem for $\mathcal{S}[q, h]$

Let q be a prime power, and $h \in \mathbb{Z}_q[x_1, x_2]$ be a symmetric polynomial. By the proof of Corollary 1 above, the problem $\mathcal{S}[q, h]$ is computationally equivalent to $Z_{\mathbf{A}}(\cdot)$, where $\mathbf{A}$ is the following $q \times q$ and q-discrete matrix:

$$A_{i,j} = \omega_q^{h(i,j)}, \qquad \text{for all } i, j \in [0 : q - 1]. \tag{13}$$

Although the Orthogonality and the Group conditions can be used to prove the #P-hardness of $\mathcal{S}[q, h]$ for many interesting polynomials h, as demonstrated in Corollary 1, it does not cover all the #P-hard $\mathcal{S}[q, h]$. For example, even if we assume that h is symmetric; and every monomial in $h(x_1, x_2)$ contains both x_1 and x_2 (and thus, $h(0, x) = h(x, 0) = 0$ for all $x \in \mathbb{Z}_q$ and the matrix $\mathbf{A}$ defined in (13) is both symmetric and *normalized*: $A_{0,i} = A_{i,0} = 1$ for all i), the Group condition can not deal with the case when there exist indices $i \neq j \in [0 : q - 1]$ such that $\mathbf{A}_{i,*} = \mathbf{A}_{j,*}$. We will use $\mathcal{C}$ to denote this class of problems.

We can prove a stronger theorem — the fourth condition. It is a *strengthening* of the current Group condition, leading to a complexity dichotomy theorem for the class $\mathcal{C}$. Due to the space limit, we omit its proof here.

Lemma 4 (The Generalized Group Condition). *Let $\mathbf{A}$ be an $m \times m$ symmetric, normalized and M-discrete matrix for some positive integer M such that for all $i, j \in [0 : m - 1]$, either $\mathbf{A}_{i,*} = \mathbf{A}_{j,*}$ or $\langle \mathbf{A}_{i,*}, \mathbf{A}_{j,*} \rangle = 0$. Let $T_1, \ldots, T_\ell$ be a partition of $[0 : m - 1]$, such that, $\mathbf{A}_{i,*} = \mathbf{A}_{j,*} \iff \exists\, k \in [\ell] : i, j \in T_k$. Then $Z_{\mathbf{A}}(\cdot)$ is #P-hard unless $\mathbf{A}$ satisfies the following* Generalized *Group condition:*

— *For all $k \in [\ell]$, $|T_k| = m/\ell$;* and *for all $i, j \in [0 : m - 1]$, there exists a $k \in [0 : m - 1]$ such that $\mathbf{A}_{k,*} = \mathbf{A}_{i,*} \circ \mathbf{A}_{j,*}$.*

By combining the Generalized Group condition with the Orthogonality condition, we are able to show that for every problem $\mathcal{S}[q,h]$ in the class $\mathcal{C}$, either $\mathcal{S}[q,h]$ is #P-hard; or we have $\mathbf{A} = \mathbf{J} \otimes \mathbf{A}'$, where $\mathbf{J}$ is an all-1 matrix and $\mathbf{A}'$ is a q-discrete unitary matrix that satisfies the original Group Condition. The latter can ultimately lead to a polynomial-time algorithm for $Z_{\mathbf{A}}(\cdot)$ and $\mathcal{S}[q,h]$, using the algorithm developed in Section 2 and 3.

Acknowledgements. We thank Eric Bach, Richard Brualdi, Michael Kowalczyk, Endre Szemeredi and Mingji Xia for helpful discussions.

References

1. Lang, S.: Algebraic Number Theory. Addison-Wesley, Reading (1970)
2. Hua, L.: Introduction to Number Theory. Springer, Heidelberg (1982)
3. Ireland, K., Rosen, M.: A Classical Introduction to Modern Number Theory. Springer, Heidelberg (1998)
4. Goldberg, L., Grohe, M., Jerrum, M., Thurley, M.: A complexity dichotomy for partition functions with mixed signs. In: Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science, pp. 493–504 (2009)
5. Cai, J.Y., Chen, X., Lu, P.: Graph homomorphisms with complex values: A dichotomy theorem. In: Proceedings of the 37th International Colloquium on Automata, Languages and Programming (2010)
6. Grabmeier, J., Kaltofen, E., Weispfenning, V.: Computer Algebra Handbook. Springer, Heidelberg (2003)
7. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, Cambridge (2003)
8. Lidl, R., Niederreiter, H.: Finite Fields. Encyclopedia of Mathematics and its Applications, vol. 20. Cambridge University Press, Cambridge (1997)
9. Ehrenfeucht, A., Karpinski, M.: The computational complexity of (XOR, AND)-counting problems. University of Bonn (1990)
10. Bulatov, A., Grohe, M.: The complexity of partition functions. Theoretical Computer Science 348(2), 148–186 (2005)
11. Cai, J.-Y., Chen, X., Lipton, R., Lu, P.: On tractable exponential sums. arXiv (1005.2632) (2010)
12. Lovász, L.: Operations with structures. Acta Mathematica Hungarica 18, 321–328 (1967)
13. Dyer, M., Greenhill, C.: The complexity of counting graph homomorphisms. In: Proceedings of the 9th International Conference on Random Structures and Algorithms, pp. 260–289 (2000)
14. Freedman, M., Lovász, L., Schrijver, A.: Reflection positivity, rank connectivity, and homomorphism of graphs. Journal of the American Mathematical Society 20, 37–51 (2007)
15. Dyer, M.E., Goldberg, L.A., Paterson, M.: On counting homomorphisms to directed acyclic graphs. Journal of the ACM 54(6) (2007)
16. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Computing 8(3), 410–421 (1979)
17. Valiant, L.: The complexity of computing the permanent. Theoretical Computer Science 8, 189–201 (1979)

Recognizing d-Interval Graphs and d-Track Interval Graphs*

Minghui Jiang

Department of Computer Science, Utah State University, Logan, UT 84322, USA
mjiang@cc.usu.edu

Abstract. A *d-interval* is the union of d disjoint intervals on the real line. A *d-track interval* is the union of d disjoint intervals on d disjoint parallel lines called tracks, one interval on each track. As generalizations of the ubiquitous interval graphs, d-interval graphs and d-track interval graphs have wide applications, traditionally to scheduling and resource allocation, and more recently to bioinformatics. In this paper, we prove that recognizing d-track interval graphs is NP-complete for any constant $d \geq 2$. This confirms a conjecture of Gyárfás and West in 1995. Previously only the complexity of the $d = 2$ case was known. Our proof in fact implies that several restricted variants of this graph recognition problem, i.e, recognizing balanced d-track interval graphs, unit d-track interval graphs, and $(2, \ldots, 2)$ d-track interval graphs, are all NP-complete. This partially answers another question recently raised by Gambette and Vialette. We also prove that recognizing depth-two 2-track interval graphs is NP-complete, even for the unit case. In sharp contrast, we present a simple linear-time algorithm for recognizing depth-two unit d-interval graphs. These and other results of ours give partial answers to a question of West and Shmoys in 1984 and a similar question of Gyárfás and West in 1995. Finally, we give the first bounds on the track number and the unit track number of a graph in terms of the number of vertices, the number of edges, and the maximum degree, and link the two numbers to the classical concepts of arboricity.

1 Introduction

A *d-interval* is the union of d disjoint (open) intervals on the real line. A *d-interval graph* is the intersection graph of a family of d-intervals. Recall that the *intersection graph* $\Omega(\mathcal{F})$ of a family of sets $\mathcal{F} = \{S_1, \ldots, S_n\}$ is the graph with $\mathcal{F}$ as the vertex set and with two different vertices S_i and S_j adjacent if and only if $S_i \cap S_j \neq \emptyset$; the family $\mathcal{F}$ is called a *representation* of the graph $\Omega(\mathcal{F})$.

Throughout the paper, the intervals are all open. The d disjoint intervals of a d-interval can be restricted in their lengths as follows. A d-interval is *balanced* if the d disjoint intervals have the same length, and is *unit* if the d disjoint intervals all have unit length. A d-interval is $(x_1, \ldots, x_d)$ if the d disjoint intervals have integer endpoints and have lengths $x_1, \ldots, x_d$, respectively, in sequential order. For a balanced d-interval graph, the d disjoint intervals that constitute the same d-interval must have the same

* Supported in part by NSF grant DBI-0743670.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 160–171, 2010.

length, although the intervals from different d-intervals may have different lengths; on the other hand, all intervals of a unit d-interval graph must have the same unit length.

The *depth* of a family of d-intervals is the maximum number of intervals that share a common point. The *representation depth* of a d-interval graph is the minimum depth of any d-interval representation of the graph. For example, any d-interval representation of a triangle-free graph must have depth at most 2, because three intervals sharing a common point would induce a triangle in the intersection graph.

A *d-track interval* is the union of d disjoint intervals on d disjoint parallel lines called *tracks*, one interval on each track. Similarly define the restrictions on d-track intervals and the corresponding variations of d-track interval graphs. The d disjoint tracks for a d-track interval graph can be viewed as d disjoint "host" intervals on the real line for a d-interval graph. Thus the class of d-track interval graphs is contained in the class of d-interval graphs. We summarize the hierarchy of d-interval graph classes:

$$(x,\ldots,x) \subseteq \text{unit} \subseteq \text{balanced} \subseteq \text{unrestricted} \qquad \text{triangle-free} \subset \text{representation depth} \leq 2$$
$$d\text{-track interval graphs} \subset d\text{-interval graphs}$$

As generalizations of the ubiquitous interval graphs, d-interval graphs and d-track interval graphs have wide applications, traditionally to scheduling and resource allocation [19,7,14,8], and more recently to bioinformatics [16,5,21,9,15,2]. A classical result of West and Shmoys [23] states that, for any constant $d \geq 2$, recognizing d-interval graphs is NP-complete (moreover, for any constants $d \geq 2$ and $r \geq 3$, recognizing d-interval graphs of representation depth at most r is also NP-complete). Recently, Gambette and Vialette [10] studied some restricted subclasses of 2-interval graphs, and observed that the lengths of 2-intervals in West and Shmoys's proof of NP-hardness of recognizing 2-interval graphs [23] can be adjusted to meet the balanced restriction. Thus recognizing balanced 2-interval graphs is NP-hard too. The complexity of recognizing the more restricted unit and (x,x) 2-interval graphs, however, was left as an open question:

Question 1 (Gambette and Vialette, 2007 [10]). What is the complexity of recognizing unit 2-interval graphs and (x,x) 2-interval graphs?

Recall that d-track interval graphs are a subclass of d-interval graphs. Gyárfás and West [13] proved that recognizing 2-track interval graphs is NP-complete (their proof also implies that for any constant $r \geq 3$, recognizing 2-track interval graphs of representation depth at most r is NP-complete), and made the following conjecture:

Conjecture 1 (Gyárfás and West, 1995 [13]). For any constant $d \geq 2$, recognizing d-track interval graphs is NP-hard.

It is easy to check that Gyárfás and West's proof of NP-hardness of recognizing 2-track interval graphs [13] can also be adapted, by adjusting the interval lengths in the representation as Gambette and Vialette [10] did for 2-interval graphs, to show that recognizing balanced 2-track interval graphs is NP-hard. But whether the proof can be adapted further to prove the NP-hardness of recognizing unit and (x,x) 2-track interval graphs, or the NP-hardness of recognizing d-track interval graphs for $d > 2$, is not at all obvious. Our main result is the following theorem that partially answers Question 1 and confirms Conjecture 1:

Theorem 1. *For any constant $d \geq 2$, recognizing d-track interval graphs, balanced d-track interval graphs, unit d-track interval graphs, and $(2, \ldots, 2)$ d-track interval graphs are all NP-complete. Moreover, for any constants $d \geq 2$ and $r \geq 3$, recognizing d-track interval graphs, balanced d-track interval graphs, unit d-track interval graphs, and $(2, \ldots, 2)$ d-track interval graphs of representation depth at most r are all NP-complete.*

West and Shmoys [23] also posed the following natural question after proving the NP-completeness of recognizing d-interval graphs of representation depth at most r for any constants $d \geq 2$ and $r \geq 3$:

Question 2 (West and Shmoys, 1984 [23]). What is the complexity of recognizing d-interval graphs of representation depth at most 2? In particular, what is the complexity of recognizing d-interval graphs that are triangle-free?

We will use the term "depth-two" as an abbreviation for "representation depth at most 2." Recall that the class of triangle-free d-interval graphs is properly contained in the class of depth-two d-interval graphs. Thus any algorithm for recognizing depth-two d-interval graphs can be augmented, by adding a straightforward step that checks the triangle-free condition, to an algorithm for recognizing triangle-free d-interval graphs. In the same spirit of Question 2 on recognizing d-interval graphs, Gyárfás and West [13] later posed the following question on recognizing 2-track interval graphs:

Question 3 (Gyárfás and West, 1995 [13]). What is the complexity of recognizing 2-track interval graphs that are triangle-free?

Our following theorem complements Theorem 1 and partially answers Question 3:

Theorem 2. *Recognizing* (i) *depth-two 2-track interval graphs and* (ii) *depth-two unit 2-track interval graphs are both NP-complete.*

We also have the following positive results that partially answer Question 2:

Theorem 3. *Let G be a graph of n vertices and m edges.* (i) *There is an $O(\text{poly}(m + n))$ time algorithm that determines, for any $d \geq 2$, either that G is not a depth-two d-interval graph, or that G is a depth-two $(d + 1)$-interval graph.* (ii) *There is an $O(2^m \text{poly}(m + n))$ time algorithm that determines, for any $d \geq 2$, whether G is a depth-two d-interval graph.* (iii) *There is an $O(m + n)$ time algorithm that determines the smallest number d such that G is a depth-two unit d-interval graph.*

The $O(\text{poly}(m + n))$ time algorithm in Theorem 3(i) implies an approximation algorithm with additive error 1 for finding the smallest number d such that G is a depth-two d-interval graph. This approximation would be best possible if it turned out that recognizing depth-two d-interval graphs is NP-hard for any constant $d \geq 2$; see Question 2. The $O(2^m \text{poly}(m + n))$ time algorithm in Theorem 3(ii) improves a previous $O(3^m(m + n))$ time algorithm by Maas [18] for recognizing depth-two d-interval graphs. Also note the sharp contrast between Theorem 2(ii) and Theorem 3(iii): it is interesting that while recognizing unrestricted 2-interval graphs and 2-track interval

Table 1. Complexities of recognizing d-interval graphs and d-track interval graphs

	d-interval graphs		d-track interval graphs	
unrestricted	NP-complete	[23]	NP-complete	Theorem 1
balanced	NP-complete ($d = 2$)	[10]	NP-complete	Theorem 1
unit	?	[10]	NP-complete	Theorem 1
$(2, \ldots, 2)$	?	[10]	NP-complete	Theorem 1
depth-two	? ($+1$ approximation)	Theorem 3(i)	NP-complete ($d = 2$)	Theorem 2
depth-two, unit	linear-time	Theorem 3(iii)	NP-complete ($d = 2$)	Theorem 2

graphs are both NP-complete, recognizing their restricted (depth-two, unit) variants are so drastically different in complexity. We summarize in Table 1 the current best results on recognizing variants of d-interval graphs and d-track interval graphs.

The graph recognition problems studied in this paper are closely related to classical problems on interval numbers and track numbers in extremal graph theory. For a graph G, the *interval number* $i(G)$ is the smallest number d such that G is a d-interval graph [20,12], and the *track number* $t(G)$ is the smallest number d such that G is a d-track interval graph [17,13]. Thus a graph is a d-interval graph (resp. d-track interval graph) if and only if its interval number (resp. track number) is at most d. Similarly define the *unit interval number* $i_u(G)$ as the smallest number d such that G is a unit d-interval graph [4], and the *unit track number* $t_u(G)$ as the smallest number d such that G is a unit d-track interval graph. It is clear that $i(G) \leq i_u(G)$, $t(G) \leq t_u(G)$, $i(G) \leq t(G)$, and $i_u(G) \leq t_u(G)$ for any graph G.

We link the track number and the unit track number of a graph to the classical concepts of arboricity. A *caterpillar* is a tree containing a dominating path such that every vertex not on the path is adjacent to some vertex on the path. A *caterpillar forest* is a graph in which every connected component is a caterpillar. The *caterpillar arboricity* $ca(G)$ of a graph G is the minimum number of caterpillar forests into which its edges can be decomposed. A *linear forest* is a graph in which every connected component is a path. The *linear arboricity* $la(G)$ of a graph G is the minimum number of linear forests into which its edges can be decomposed. It is clear that $ca(G) \leq la(G)$ for any graph G. Akiyama, Exoo, and Harary [1] conjectured that every graph G of maximum degree Δ satisfies $la(G) \leq \lceil (\Delta + 1)/2 \rceil$. This conjecture has been shown to be correct asymptotically as $\Delta \to \infty$ [3], and has been confirmed for graphs of small constant degrees [1].

Let G be a graph of n vertices, m edges, and maximum degree Δ. It is known that $i(G) \leq \lceil (n+1)/4 \rceil$ [11], $i_u(G) \leq \lceil (n-1)/2 \rceil$ [4], $i(G) \leq \lceil \sqrt{m}/2 \rceil + 1$ [6], and $i(G) \leq i_u(G) \leq \lceil (\Delta + 1)/2 \rceil$ [12,22]. These bounds are best possible. Our following theorem gives complementary bounds of these types:

Theorem 4. *Let G be a graph of n vertices, m edges, and maximum degree Δ. Then* (i) $t(G) \leq \lfloor n/2 \rfloor$; (ii) $t_u(G) \leq \lceil m/2 \rceil$, *and this bound is best possible;* (iii) $t(G) \leq ca(G)$ *and* $t_u(G) \leq la(G)$. *In particular, if* $la(G) \leq \lceil (\Delta + 1)/2 \rceil$ *as conjectured, then* $t(G) \leq t_u(G) \leq \lceil (\Delta + 1)/2 \rceil \leq \lceil n/2 \rceil$ *and* $t(G) \leq \lceil \sqrt{m} \rceil$.

2 Hardness of Recognizing d-Track Interval Graphs

In this section we prove Theorem 1. The most basic part of our result, which confirms Conjecture 1 of Gyárfás and West [13], is that recognizing d-track interval graphs for any constant $d \geq 2$ (RDT) is NP-hard. We will show that RDT is NP-hard by a reduction from the following NP-hard problem [23, 13]:

> *Hamiltonian path in triangle-free cubic graph (HP3):* Given a triangle-free cubic graph $G = (V, E)$ and an edge $uv \in E$, decide whether there is a Hamiltonian path in G starting at u and ending at v.

In the following, we first study the simple case of $d = 2$ and prove that recognizing 2-track interval graphs (R2T) is NP-hard, then turn to the general case and prove that recognizing d-track interval graphs for any constant $d \geq 2$ (RDT) is NP-hard. Our proof, although presented in a progressive way, will reduce HP3 to both R2T and RDT directly; indeed the reduction to R2T is exactly the same as the reduction to RDT with $d = 2$. In contrast, West and Shmoys's proof for d-interval graphs [23] first reduces HP3 to recognizing 2-interval graphs, then inductively reduces recognizing $(d-1)$-interval graphs to recognizing d-interval graphs.

In any d-track interval representation of a graph, each edge uv of the graph has to be *realized* on at least one of the d tracks, that is, represented by two overlapping intervals, one interval of u and one interval of v, on one of the d tracks. If the intervals of u and v overlap on exactly one of the d tracks, then the edge uv is said to be *uniquely realized* on that track. A d-track interval representation of a graph is *continuous* on some track if the union of all intervals on that track is a single interval (with no holes).

Lemma 1. *In any multiple-interval representation of a graph, if the depth of the representation is at most 2, then at most $t-1$ edges of the graph can be realized by t intervals. Moreover, if exactly $t-1$ edges are realized by t intervals, then the representation of the t intervals must be continuous.*

Lemma 2. *$K_{2d,2d-1}$ has a continuous $(2, \ldots, 2)$ d-track interval representation (Fig. 1). Every d-track interval representation of $K_{2d,2d-1}$ must be continuous on each track.*

2.1 R2T Is NP-Hard

Given a triangle-free cubic graph $G = (V, E)$ with the vertex set $V = \{v_1, \ldots, v_n\}$ and an edge $v_1v_n \in E$, we will construct an extended graph G_2 such that G has a Hamiltonian path between v_1 and v_n if and only if G_2 has a 2-track interval representation. Note that G has exactly $3n/2$ edges, so the number n of vertices must be even.

We refer to Fig. 2 for the construction of the extended graph G_2. The following subgraphs are used in our construction:

1. A graph $\ddot{G}$ obtained from G by deleting the edge v_1v_n;
2. An independent set Q of n vertices q_i, $1 \leq i \leq n$;

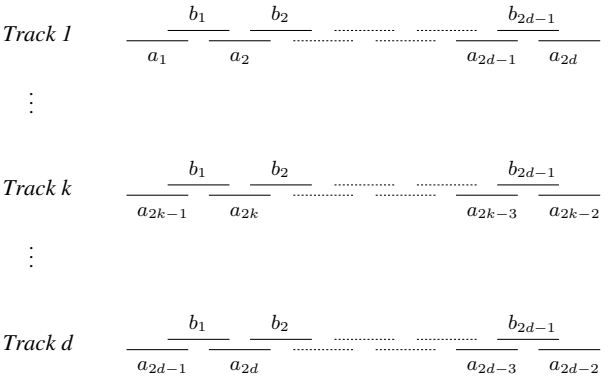


Fig. 1. A continuous d-track interval representation of the complete bipartite graph $K_{2d,2d-1}$

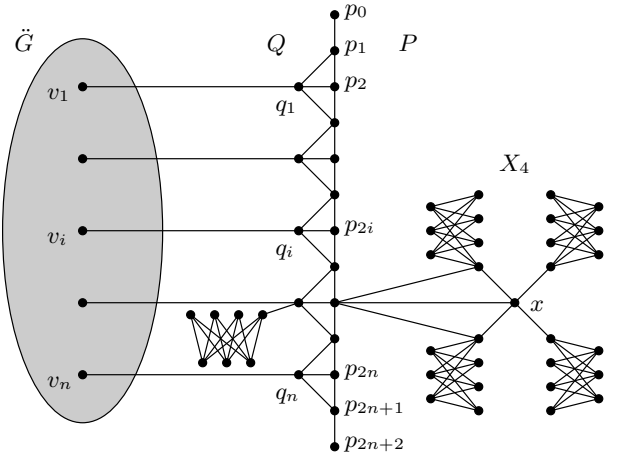


Fig. 2. The extended graph G_2. Each vertex q_i in the independent set Q is connected to a distinct copy of the complete bipartite graph $K_{4,3}$ (only one copy is shown here). Each vertex p_j on the path P is connected to a distinct copy of the gadget X_4 (only one copy is shown here).

3. A path $P = \langle p_0 p_1 \dots p_{2n+2} \rangle$ of $2n + 3$ vertices p_j, $0 \le j \le 2n + 2$;
4. The complete bipartite graph $K_{4,3}$;
5. A gadget X_4 consisting of a vertex x and four copies of $K_{4,3}$, where x is connected to a vertex of degree 3 in each copy of $K_{4,3}$.

The extended graph G_2 is composed by connecting the graph $\ddot{G}$, the independent set Q, the path P, n copies of the complete bipartite graph $K_{4,3}$, and $2n + 3$ copies of the gadget X_4 as follows:

- Connect $\ddot{G}$ to Q by the n edges $v_i q_i$, $1 \le i \le n$.
- Connect Q to P by $3n$ edges: three edges from each vertex q_i, $1 \le i \le n$, to the three vertices p_{2i-1}, p_{2i}, and p_{2i+1}.

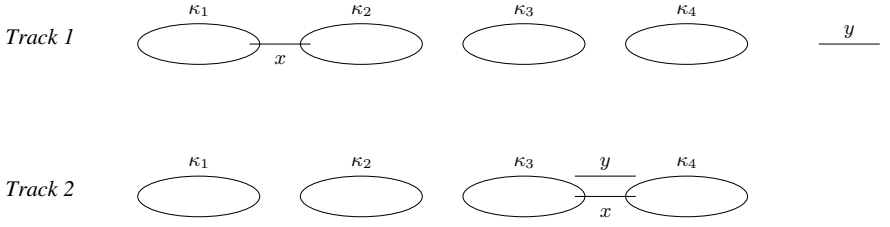


Fig. 3. A 2-track interval representation of the graph $X_4(y)$. Each of the four copies κ_1, κ_2, κ_3, and κ_4 of $K_{4,3}$ in X_4 is represented continuously (shown schematically as ovals), following the pattern in Fig. 1 with $d = 2$.

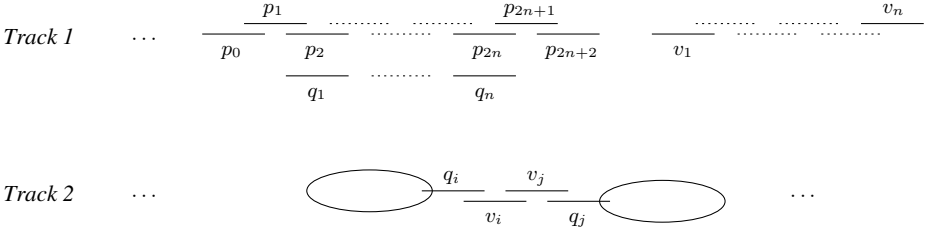


Fig. 4. A 2-track interval representation of the extended graph G_2

- Connect each vertex q_i in Q to a vertex of degree 3 in a distinct copy of $K_{4,3}$.
- Connect each vertex p_j in P to a distinct copy of X_4: three edges from p_j to the vertex x and two neighbors of x in X_4.

Denote by $X_4(y)$ the graph consisting of a vertex y and a copy of the gadget X_4, where y is connected to the vertex x and two neighbors of x in X_4. Then each vertex p_j and the corresponding copy of the gadget X_4 induce a subgraph $X_4(p_j)$ in G_2. The following lemma shows an important property of these subgraphs.

Lemma 3. *The graph $X_4(y)$ has a $(2, 2)$ 2-track interval representation (Fig. 3). Moreover, in any 2-track interval representation of $X_4(y)$, there is a track on which the interval of y is completely covered by the intervals of X_4.*

Since each vertex p_j is connected to a distinct copy of X_4, Lemma 3 implies that at most one interval of each vertex p_j is *free*, that is, at most one interval of p_j on one of the d tracks is available to overlap with the intervals of other vertices of P and the vertices of Q.

We now show that the input graph G has a Hamiltonian path between v_1 and v_n if and only if the extended graph G_2 has a 2-track interval representation. Refer to Fig. 4. We first prove the "only if" direction. Suppose that G has a Hamiltonian path $\langle v_1 \dots v_n \rangle$. Then the edges of G that are not on the Hamiltonian path form a perfect matching in G. Construct a 2-track interval representation of the extended graph G_2 as follows:

- Realize the path $P = \langle p_0 p_1 \dots p_{2n+2} \rangle$ on track 1 by consecutively overlapping intervals. Realize the edges between Q and P also on track 1, by choosing the interval of q_i the same as the interval of p_{2i}.

- Realize the Hamiltonian path $\langle v_1 \dots v_n \rangle$ on track 1 by consecutively overlapping intervals.
- Realize each edge $v_i v_j$ of $\ddot{G}$ that is not on the Hamiltonian path and the two edges $v_i q_i$ and $v_j q_j$ on track 2 by disjoint groups of four intervals.
- Realize the edge between each vertex q_j in Q and the corresponding copy of $K_{4,3}$ on track 2.
- Realize each subgraph $X_4(p_j)$ following the pattern in Fig. 3 with $y = p_j$ such that the one free interval of each p_j is on track 1 (the path P and the edges between Q and P are realized by these free intervals).

Note that this representation of G_2 can be easily adapted to a $(2, 2)$ 2-track interval representation. This completes the "only if" direction of the proof.

We next prove the "if" direction. Suppose that G_2 has a 2-track interval representation. We will find a Hamiltonian path between v_1 and v_n in G. By Lemma 3, at most one interval of each vertex p_j is free. Thus all edges of the path $\langle p_0 p_1 \dots p_{2n+2} \rangle$ must be uniquely realized on the same track, say track 1, by consecutively overlapping intervals. Then all edges between Q and P must be uniquely realized on track 1 too. Since the interval of each vertex q_i is completely covered by the three consecutively overlapping intervals of p_{2i-1}, p_{2i}, and p_{2i+1} on track 1, the other two edges incident to q_i (the edge $v_i q_i$ and the edge between q_i and the corresponding copy of $K_{4,3}$) must be uniquely realized on track 2.

By Lemma 2, the representation of $K_{4,3}$ is continuous on each track. Since q_i is adjacent to only one vertex of $K_{4,3}$, at least one endpoint of the interval of q_i must be covered by the union of the intervals of $K_{4,3}$ on track 2. Similarly, since v_i is adjacent to q_i but not to $K_{4,3}$, at least one endpoint of the interval of v_i must be covered by the interval of q_i on track 2. For each edge $v_i v_j$ of G, if the two intervals of v_i and v_j overlap on track 2, then they must overlap only at their endpoints that are not covered by the intervals of q_i and q_j. Since no three vertices of the triangle-free graph G can have three overlapping intervals that share a common point, neither the interval of v_i nor the interval v_j can overlap with the interval of a third vertex v_k on track 2. That is, for each vertex v_i of G, at most one edge of G incident to v_i can be realized on track 2. Since each edge is incident to two vertices, it follows that at most $n/2$ edges of G can be realized on track 2.

Recall that the cubic graph G has n vertices and $3n/2$ edges, and that $\ddot{G}$ is obtained from G by deleting the edge $v_1 v_n$. Thus at least $3n/2 - 1 - n/2 = n - 1$ edges of $\ddot{G}$ must be realized on track 1, by n intervals. Then, by Lemma 1, *exactly* $n - 1$ edges of $\ddot{G}$ must be realized by n *continuous* intervals on track 1. It follows that *exactly* $n/2$ edges of $\ddot{G}$ must be realized on track 2, one edge incident to each vertex v_i. Thus the $n - 1$ edges of $\ddot{G}$ realized on track 1 must include exactly two edges incident to each vertex v_i other than v_1 and v_n, and exactly one edge incident to each of the two special vertices v_1 and v_n. A connected graph of n vertices and $n - 1$ edges is a tree, and a tree of maximum degree 2 is a path. Therefore these $n - 1$ edges, which are realized by n continuous intervals, form a Hamiltonian path between v_1 and v_n. This completes the "if" direction of the proof.

2.2 RDT Is NP-Hard

Given a triangle-free cubic graph $G = (V, E)$ with the vertex set $V = \{v_1, \ldots, v_n\}$ and an edge $v_1v_n \in E$, we will construct an extended graph G_d such that G has a Hamiltonian path between v_1 and v_n if and only if G_d has a d-track interval representation. We refer back to Fig. 2. Similar to the construction of G_2, the construction of G_d uses the following subgraphs:

1. A graph $\ddot{G}$ obtained from G by deleting the edge v_1v_n;
2. An independent set Q of n vertices q_i, $1 \leq i \leq n$;
3. A path $P = \langle p_0p_1 \ldots p_{2n+2} \rangle$ of $2n + 3$ vertices p_j, $0 \leq j \leq 2n + 2$;
4. The complete bipartite graph $K_{2d,2d-1}$;
5. A gadget X_{2d} consisting of a vertex x and $2d$ copies of $K_{2d,2d-1}$, where x is connected to a vertex of degree $2d - 1$ in each copy of $K_{2d,2d-1}$.

Note that the graph $\ddot{G}$, the independent set Q, and the path P here are the same as those for the $d = 2$ case, but the complete bipartite graph is generalized from $K_{4,3}$ to $K_{2d,2d-1}$, and the gadget is generalized from X_4 to X_{2d}.

The extended graph G_d is composed by connecting the graph $\ddot{G}$, the independent set Q, $d - 1$ copies of the path P, n copies of the complete bipartite graph $K_{2d,2d-1}$, and $(d - 1)^2(2n + 3) + (d - 2)n$ copies of the gadget X_{2d} as follows:

- Connect $\ddot{G}$ to Q by the n edges v_iq_i, $1 \leq i \leq n$.
- Connect Q to each of the $d - 1$ copies of P by $3n$ edges: three edges from each vertex q_i, $1 \leq i \leq n$, to the three vertices p_{2i-1}, p_{2i}, and p_{2i+1}.
- Connect each vertex q_i in Q to a vertex of degree $2d - 1$ in a distinct copy of $K_{2d,2d-1}$.
- Connect each vertex p_j in each of the $d - 1$ copies of P to $d - 1$ distinct copies of X_{2d}: three edges from p_j to the vertex x and two neighbors of x in each copy of X_{2d}.
- Connect each vertex v_i in $\ddot{G}$ to $d - 2$ distinct copies of X_{2d}: three edges from v_i to the vertex x and two neighbors of x in each copy of X_{2d}.

Note that the graph G_d, when $d = 2$, is exactly the same as the graph G_2 in the previous subsection.

Denote by $X_{2d}^k(y)$ the graph consisting of a vertex y and k copies of the gadget X_{2d}, where y is connected to the vertex x and two neighbors of x in each copy of X_{2d}. The extended graph G_d contains the n subgraphs $X_{2d}^{d-2}(v_i)$, $1 \leq i \leq n$, and $d - 1$ copies of each subgraph $X_{2d}^{d-1}(p_j)$, $0 \leq j \leq 2n+2$. The following lemma generalizes Lemma 3:

Lemma 4. *Let $1 \leq k \leq d$. The graph $X_{2d}^k(y)$ has a $(2, \ldots, 2)$ d-track interval representation. Moreover, in any d-track interval representation of $X_{2d}^k(y)$, there are k tracks such that on each of the k tracks the interval of y is completely covered by the intervals of X_{2d}.*

A consequence of Lemma 4 is that each vertex v_i has at most two free intervals, and each copy of the vertex p_i has at most one free interval. Then by a similar argument as in the $d = 2$ case, we can show that the input graph G has a Hamiltonian path between v_1 and v_n if and only if the extended graph G_d has a d-track interval representation.

3 Algorithm for Recognizing Depth-Two d-Interval Graphs

In this section we prove part (i) of Theorem 3. Let $G = (V, E)$ be a graph of n vertices and m edges. Observe that in any depth-two d-interval representation of the graph, each edge uv corresponds to at least one pair of overlapping intervals of the two vertices u and v: either the two intervals overlap partially, or one interval contains the other. In either case, the intersection of the two intervals do not overlap with the other intervals. We will use the following lemma by West [22] (see also [12]):

Lemma 5 (West, 1989 [22]). *Every graph has a multiple-interval representation of depth at most 2 in which each vertex v is represented by at most $\lfloor(\deg(v) + 1)/2\rfloor$ unit intervals, except for an arbitrarily specified vertex w that appears left-most in the representation and is represented by at most $\lceil(\deg(w) + 1)/2\rceil$ unit intervals, where $\deg(v)$ denotes the degree of a vertex v.*

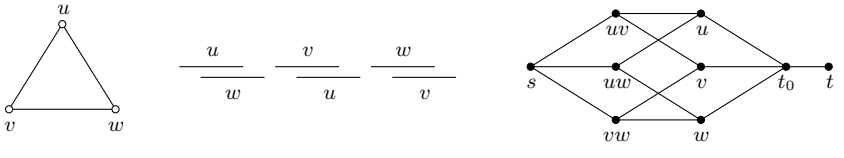
We now present a polynomial-time algorithm that determines, for any $d \geq 2$, either that G is not a depth-two d-interval graph, or that G is a depth-two $(d + 1)$-interval graph. We will reduce the graph recognition problem to a maximum flow problem on a network G' constructed from the graph G. The network G' consists of $n' = m + n + 3$ nodes: the m edges in E, the n vertices in V, the source s, the sink t, and the pre-sink t_0. These nodes are connected by $m' = 3m + n + 1$ arcs:

- One arc (s, e) of capacity 2 from the source s to each edge e in E.
- Two arcs (e, u) and (e, v), each of capacity 2, from each edge e in E to its two incident vertices u and v in V.
- One arc (v, t_0) of capacity $2d$ from each vertex v in V to the pre-sink t_0.
- One arc (t_0, t) of capacity $\min\{2m, 2dn - 1\}$ from the pre-sink t_0 to the sink t.

We refer to Fig. 5 for an example of the construction. The following two lemmas establish the relation between the graph G and the network G':

Lemma 6. *If G has a d-interval representation of depth at most 2, then G' has a flow of value $2m$.*

Proof. Consider any depth-two d-interval representation of G. Assume without loss of generality that the $2dn$ endpoints of the dn intervals are all distinct (this can be achieved by a standard procedure for interval graphs). For each edge uv, find in the representation



(a) (b) (c)

Fig. 5. An example of the reduction from graph recognition to maximum flow. (a) The graph G. (b) A 2-interval representation of depth 2 for G. (c) The flow network G'.

any pair of overlapping intervals I_u and I_v of the two vertices u and v, respectively, then assign the two endpoints of the intersection $I_u \cap I_v$ to the edge uv. There are three cases: if $I_u \subset I_v$, then both endpoints of I_u are assigned; if $I_v \subset I_u$, then both endpoints of I_v are assigned; otherwise I_u and I_v properly intersect, and one endpoint of each interval is assigned. Since the representation has depth at most 2, each edge of G is assigned two distinct endpoints from the $2dn$ interval endpoints in the representation.

We now construct a flow f of value $2m$ in the network G'. First, for each edge e, set $f(s,e) = 2$. Next, for each edge e and for each vertex v incident to e, set $f(e,v)$ to the number of endpoints of v assigned to e (either 0, 1 or 2). Then, for each vertex v, set $f(v,t_0) = \sum_e f(e,v)$, where e ranges over all edges incident to v. Finally, set $f(t_0,t) = \sum_v f(v,t_0)$, which is clearly $2m$. By Lemma 1, at most $dn-1$ edges can be realized by dn intervals. Thus $2m \le 2(dn-1) < 2dn-1$. □

Lemma 7. *If G' has a flow of value $2m$, then G has a $(d+1)$-interval representation of depth at most 2.*

Proof. Let f be a flow of value $2m$ in G'. Since all arc capacities are integers, we can assume that f is integral. The flow f is maximum since the total capacity of the arcs from the source is $2m$. In particular, this implies that $f(s,e) = 2$ for every edge e. Obtain a partition $E = E_1 \cup E_2$ of the edges such that each edge $e = uv$ belongs to E_1 if $f(e,u) = f(e,v) = 1$, and belongs to E_2 otherwise. For each vertex v, let $d_2(v)$ be the number of edges e incident to v such that $f(e,v) = 2$. Then $\sum_v d_2(v) = |E_2|$. To construct a $(d+1)$-interval representation of depth at most 2 of the graph G, we will first use $d - d_2(v)$ intervals for each vertex v to realize the edges in E_1, then use $1 + d_2(v)$ intervals for each vertex v to realize the edges in E_2.

We first consider the edges in E_1. Since the capacity of each arc (v,t_0) is $2d$, we have $f(v,t_0) \le 2d$ for each vertex v. Thus each vertex v is incident to at most $2d - 2d_2(v)$ edges in E_1. Moreover, since the capacity $\min\{2m, 2dn-1\}$ of the arc (t_0,t) is less than the total capacity $2dn$ of the n arcs (v,t_0), at least one vertex, say w, must have $f(w,t_0) \le 2d-1$. Thus w is incident to at most $2d-1-2d_2(w)$ edges in E_1. Note that

$$\lfloor (2d - 2d_2(v) + 1)/2 \rfloor = d - d_2(v) \quad \text{and} \quad \lceil (2d - 1 - 2d_2(w) + 1)/2 \rceil = d - d_2(w).$$

Thus, by Lemma 5, the subgraph $G_1 = (V, E_1)$ has a multiple-interval representation of depth at most 2 in which each vertex v is represented by at most $d - d_2(v)$ intervals. Add dummy intervals until each vertex v is represented by exactly $d - d_2(v)$ intervals.

We next consider the edges in E_2. Add n disjoint long intervals to the representation, one for each vertex. Then, for each edge e incident to a vertex v such that $f(e,v) = 2$, add a short interval I_v for v to the representation such that I_v is contained in the long interval of the other vertex of e, and does not intersect any other intervals. Thus, using one long interval and $d_2(v)$ short intervals for each vertex v, all edges in E_2 are realized. Altogether, we have exactly $d - d_2(v) + 1 + d_2(v) = d + 1$ intervals for each vertex v in the representation. □

References

1. Akiyama, J., Exoo, G., Harary, F.: Covering and packing in graphs III: cyclic and acyclic invariants. Mathematica Slovaca 30, 405–417 (1980)
2. Alcón, L., Cerioli, M.R., de Figueiredo, C.M.H., Gutierrez, M., Meidanis, J.: Tree loop graphs. Discrete Applied Mathematics 155, 686–694 (2007)
3. Alon, N.: The linear arboricity of graphs. Israel Journal of Mathematics 62, 311–325 (1988)
4. Andreae, T.: On the unit interval number of a graph. Discrete Applied Mathematics 22, 1–7 (1988)
5. Bafna, V., Narayanan, B., Ravi, R.: Nonoverlapping local alignments (weighted independent sets of axis-parallel rectangles). Discrete Applied Mathematics 71, 41–53 (1996)
6. Balogh, J., Pluhár, A.: A sharp edge bound on the interval number of a graph. Journal of Graph Theory 32, 153–159 (1999)
7. Bar-Yehuda, R., Halldórsson, M.M., Naor, J.(S.), Shachnai, H., Shapira, I.: Scheduling split intervals. SIAM Journal on Computing 36, 1–15 (2006)
8. Butman, A., Hermelin, D., Lewenstein, M., Rawitz, D.: Optimization problems in multiple-interval graphs. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 268–277 (2007)
9. Crochemore, M., Hermelin, D., Landau, G.M., Rawitz, D., Vialette, S.: Approximating the 2-interval pattern problem. Theoretical Computer Science 395, 283–297 (2008)
10. Gambette, P., Vialette, S.: On restrictions of balanced 2-interval graphs. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 55–65. Springer, Heidelberg (2007)
11. Griggs, J.R.: Extremal values of the interval number of a graph, II. Discrete Mathematics 28, 37–47 (1979)
12. Griggs, J.R., West, D.B.: Extremal values of the interval number of a graph. SIAM Journal on Algebraic and Discrete Methods 1, 1–7 (1980)
13. Gyárfás, A., West, D.B.: Multitrack interval graphs. Congressus Numerantium 109, 109–116 (1995)
14. Halldórsson, M.M., Karlsson, R.K.: Strip graphs: recognition and scheduling. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 137–146. Springer, Heidelberg (2006)
15. Jiang, M.: Approximation algorithms for predicting RNA secondary structures with arbitrary pseudoknots. IEEE/ACM Transactions on Computational Biology and Bioinformatics 7, 323–332 (2010)
16. Joseph, D., Meidanis, J., Tiwari, P.: Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In: Nurmi, O., Ukkonen, E. (eds.) SWAT 1992. LNCS, vol. 621, pp. 326–337. Springer, Heidelberg (1992)
17. Kumar, N., Deo, N.: Multidimensional interval graphs. Congressus Numerantium 102, 45–56 (1994)
18. Maas, C.: Determining the interval number of a triangle-free graph. Computing 31, 347–354 (1983)
19. Roberts, F.S.: Graph Theory and Its Applications to Problems of Society. SIAM, Philadelphia (1987)
20. Trotter Jr., W.T., Harary, F.: On double and multiple interval graphs. Journal of Graph Theory 3, 205–211 (1979)
21. Vialette, S.: On the computational complexity of 2-interval pattern matching problems. Theoretical Computer Science 312, 223–249 (2004)
22. West, D.B.: A short proof of the degree bound for interval number. Discrete Mathematics 73, 309–310 (1989)
23. West, D.B., Shmoys, D.B.: Recognizing graphs with fixed interval number is NP-complete. Discrete Applied Mathematics 8, 295–305 (1984)

Categorial Semantics of a Solution to Distributed Dining Philosophers Problem*

Zhen You[1,2], Jinyun Xue[1,2], and Shi Ying[1]

[1] State Key Lab of Software Engineering, Wuhan University, 430072, Wuhan, Hubei Province, P.R.China
[2] Provincial Key Laboratory for High-Performance Computing Technology, Jiangxi Normal University, 330022, Nanchang, Jiangxi Province, P.R.China

Abstract. Distributed dining philosophers is regarded as one of the most representative resource allocation problems. Many strategies are employed for avoiding deadlock and starvation, the two well-known problems in Distributed Dining Philosophers Problem(DDPP). In this paper, the formal semantics of DDPP are originally proposed by using category theory based on the Chandy-Mirsa's acyclic directed graph strategy. The goal is to demonstrate how category theory is used in precisely defining categorical semantics and diagrammatically describing philosophers' priority, states-transition, and composition of processes, rather than to design a new algorithm to solve the DDPP. Compared with other formal techniques, category theory not only provides a good mathematical structure for formalizing different relationships and interactions at different abstract levels, but also its diagrammatical representation strengthens the traceability and understandability of philosophers' priority and states-transformation; additionally, its universal constructions (like colimit) offer the ability to manipulate and reason about system configuration.

Keywords: Categorial Semantics, Distributed Dining Philosophers Problem, Category Theory, Universal Constructions.

1 Introduction

The resource allocation problem is generally considered as the fundamental problem in distributed systems. Dining philosophers is a one of the most typical distributed resource allocation problems. In 1965, Edsger Dijkstra set an examination question on a synchronization problem where five computers competed for access to five shared tape drive peripherals. Soon the problem was retold by Tony Hoare as the Dining Philosophers Problem (DPP).

The ***Classical Dining Philosophers Problem(CDPP)*** was originally introduced for a ring topology by Dijkstra [1]. Five philosophers are sitting around

* This research is supported by the NSFC Grant No. 60573080 and 60773054, and International Cooperation Project of the Ministry of Science and Technology of China, Grant No. 2008DFA11940.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 172–184, 2010.

a circular table. Each philosopher has his own place, a single fork between each pair of adjacent philosophers. Any philosopher may decide to eat at any time and requires both of his forks to do so. The ***Distributed Dining Philosophers Problem(DDPP)***[2], also called as the general philosophers problem, states there are n symmetrical philosophers (with identical protocols) contending for access to m resources. Each philosopher requires a fixed subset of the resources and has to acquire all required resources for the philosopher to perform its task (such as eating), and every resource could be allocated to at most one philosopher at a time. The CDPP is a special case of DDPP.

Deadlock and starvation are the most serious phenomena, which should be avoided in tackling the resource allocation problem. ***Deadlock*** occurs when two or more processes in a system are blocked forever, because of requirements that can never be satisfied. ***Starvation*** happens when one or more processes can never use resources.

Theretofore, several attempts have been made to find a satisfactory solution to the dining philosophers problem in a distributed environment. The queue of waiting neighbors [3],[4] and token-passing scheme are general approaches to avoid deadlock and starvation, but result in "strict mutual exclusion problem". Probabilistic algorithm with fairness [5] and one without fairness [6] can also be used, but it's difficult to verify the correctness. Formal models (e.g. STOCS [7], SPANNER [8]) and automated verification tools (e.g. ARC [9], PVS [10]) are also applied to solve this problem. A famous solution for DDPP is based on an acyclic directed graph technique which was firstly presented by K. M. Chandy and J. Misra [11]. This strategy can be efficiently guarantee safety, liveness and fairness of dining philosophers problem as well as drinking philosophers problem.

However, there still is no effective methodology to defining the formal semantics of DDPP by using category theory. The motivation of this paper is not to propose new algorithm to solve the DDPP, but to illustrate how category theory is used in describing formal semantics of a particular solution of DDPP, which based upon the Chandy-Misra's acyclic directed graph strategy [11]. The diagrammatical representation and powerful expressive ability of category theory make it easier to clarify the traceability and understandability of DDPP.

The rest of this paper is organized as follows. Section 2 reviews some properties of DDPP and foundational definitions of category theory; Chandy-Misra's solution to DDPP is briefly presented in Section 3; The next Section detailedly demonstrates the categorial semantics of DDPP from four different levels; concluding remarks and future works are finally discussed in Section 5.

2 Background Knowledge

2.1 Properties of DDPP

Each philosopher has 3 phases: thinking, hungry, and eating. Associated with a philosopher process i is a variable "$i.phase$" that takes on variable "t, h, e".

(1)Thinking: The process does not require any resource. We employ a boolean variable $i.t \equiv (i.phase = t)$. A process spontaneously makes transition from thinking to hungry phase in finite time.

(2)Hungry: The process requires all resources adjacent to it. We employ a boolean variable $i.h \equiv (i.phase = h)$.

(3)Eating: When all resources required by a process are acquired, its phase is changed to eating. We employ a boolean variable $i.e \equiv (i.phase = e)$.

A solution to DDPP guarantees each philosopher is eventually able to enter its critical section and access its needed resources with satisfying the following properties:

Safety: Neighbors never eat simultaneously.

Progress/Liveness: Every hungry process eventually eats.

The safety property is weaker than "strict mutual exclusion", which doesn't allow two processes in the system, no matter how far apart, to enter the critical section simultaneously. Here, only two adjacent neighbors can't eat at the same time; we permit the non-neighbors to eat simultaneously. The deadlock and starvation can be avoided if the system meet the restriction of progress.

2.2 Category Theory

As a relatively young branch of mathematics, category theory, which is a formal tool similar to set theory, is designed to demonstrate various structural concepts of different fields in a uniform way. In recent years, several researchers indicated that category theory can be used for formalizing many aspects in computer science, especially software engineering, including the design and implementation of programming languages[12],[13], models of concurrency[14],[15] type theory[16],[17], specification language[18], automata theory[19],[20], architecture[21],[22], and syntax and semantic model [23],[24].

Some fundamental definitions [25],[26] needed in the rest of paper are given.

Definition 1. ***Category***
A category $\mathcal{C}$ is given by a collection $\mathcal{C}_0$ of objects and a collection $\mathcal{C}_1$ of morphisms which have the following structure.

- Each morphism in $\mathcal{C}_1$ has a ***domain*** and a ***codomain***, which are objects in $\mathcal{C}_0$; one writes $f : \mathcal{X} \to \mathcal{Y}$ if $\mathcal{X}$ is the domain of the morphism f, and $\mathcal{Y}$ its codomain. One also writes $\mathcal{X} = dom(f)$ and $\mathcal{Y} = cod(f)$;
- Given two morphisms $f : \mathcal{X} \to \mathcal{Y}$ and $g : \mathcal{Y} \to \mathcal{Z}$ such that $cod(f) = dom(g)$, the composition of f and g, written $g \circ f : \mathcal{X} \to \mathcal{Z}$ (called the ***composition law***), is defined and has domain $dom(f)$ and codomain $cod(g)$;
- Composition is associative, that is: given $f : \mathcal{X} \to \mathcal{Y}$, $g : \mathcal{Y} \to \mathcal{Z}$ and $h : \mathcal{Z} \to \mathcal{W}$, $h \circ (g \circ f) = (h \circ g) \circ f$ (called the ***associative law***);
- For every object $\mathcal{X}$ there is an identity morphism $id_{\mathcal{X}} : \mathcal{X} \to \mathcal{X}$, satisfying $id_{\mathcal{X}} \circ g = g$ for every $g : \mathcal{Y} \to \mathcal{X}$ and $f \circ id_{\mathcal{X}} = f$ for every $f : \mathcal{X} \to \mathcal{Y}$ (called the ***identity law***).

Definition 2. ***Graph homomorphism***
Let $\mathcal{G}$ and $\mathcal{H}$ be graphs. A homomorphism of graphs $\varphi : \mathcal{G} \to \mathcal{H}$ is a pair of maps $\varphi_0 : \mathcal{G}_0 \to \mathcal{H}_0$ and $\varphi_1 : \mathcal{G}_1 \to \mathcal{H}_1$ such that for each arrow $f : x \to y$ of $\mathcal{G}$ we have $\varphi_1(f) : \varphi_0(x) \to \varphi_0(y)$ in $\mathcal{H}$.

Definition 3. ***Diagram***
Let $\mathcal{C}$ be a category and $\mathcal{I}$ a graph. A diagram in $\mathcal{C}$ with shape $\mathcal{I}$ is a graph homomorphism $\delta : \mathcal{I} \to graph(\mathcal{C})$.

Definition 4. ***Commutative Diagrams***
Let $\mathcal{C}$ be a category, its diagram $\mathcal{D}$ is said to commute iff, for every pair x,y of nodes and every pair of paths $W = U_1, U_2, ..., U_m$, $W' = V_1, V_2, ..., V_n$ from x to y in diagram $\mathcal{D}$,

$$U_m \circ U_{m-1} \circ ... \circ U_1 = V_n \circ V_{n-1} \circ ... \circ V_1$$

holds in category $\mathcal{C}$.

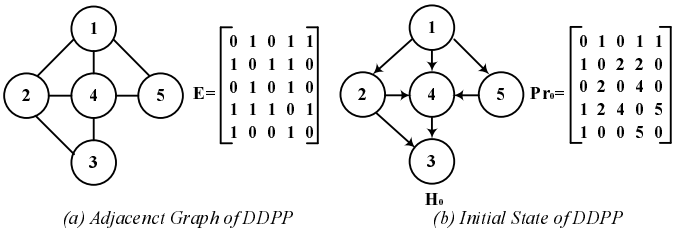
Definition 5. ***Colimit***
Let $\delta : \mathcal{I} \to graph(\mathcal{C})$ be a diagram in category $\mathcal{C}$. A colimit of δ is a commutative cocone $p : \delta \to z$ such that, for every other commutative cocone $p' : \delta' \to z'$, there is a unique morphism $f : z \to z'$ such that $f \circ p = p'$.

Remark 1. Colimit is a generalization of other universal constructions, which includes initial object, sum(coproduct), pushout and coequalizer.

3 Acyclic Graphic Solution of DDPP

In this section, we briefly present Chandy-Mirsa's acyclic directed graph technique for solving the DDPP. Firstly, DDPP with n symmetrical philosophers and m resources could be represented by a ***static, finite and undirected graph*** G with n nodes and m edges [27]. The nodes of G represent philosophers, and the edge between node i and node j is denoted by (i, j), is equivalently by (j, i), which means philosopher i and philosopher j shares a resource. For convenience, the graph can also be represented by the ***constant integer matrix*** E with n rows and n columns, where $E(i, j) = 1$ holds if and only if there is an edge between i and j in graph G, otherwise $E(i, j) = 0$. Since G is undirected, $E(i, j) = E(j, i)$. There is no edge in G from a node to itself, $E(i, i) = 0$, for all node.

Example: As the picture (a) in Fig. 1 illustrated, a DDPP with 5 philosophers and 7 resources could be described by graph G and integer matrix E.
Deadlock, firstly recognized and analyzed in 1968 by E. W. Dijkstra [32], who termed it as ***deadly embrace***, can cause an indefinite circular wait among some processes. The solution of deadlock requires the selection of one process -"the winner"- from a set of conflicting processes by imposing an asymmetry on processes that express the different precedence. The priority between pairs of potentially conflicting processes could be depicted by a ***directed graph*** H, obtained from its homomorphism graph G by giving the directions to edges in G as follows: An edge in H is directed from the process with greater priority toward the process with lesser priority. Besides the direction of edges, the priority can also be represented by the ***dynamic integer matrix*** Pr with n rows and n



(a) Adjacenct Graph of DDPP *(b) Initial State of DDPP*

Fig. 1. Graphic Description of DDPP

columns, where $Pr(i,j) = i$ if and only if the edge is directed from i to j in graph H, we also say i has priority over j. Of course, $Pr(i,j) = Pr(j,i)$.

Meanwhile, we assume that the DDPP's initial state is described as an ***acyclic directed Graph*** H_0 and the ***initial matrix*** Pr_0 in the picture (b) of Fig. 1. The next states of DDPP are H_i $(i > 0)$.

Some models of concurrency involve synchronous message passing (e.g. CCS[28] and CSP[29]), and some utilize temporal logic languages (e.g. Tempura[30] and PTL[31]) with shared variables to tackle communication between concurrent processes. The following conditions and theorems are depicted by using UNITY temporal logic [27].

When a hungry philosopher i intends to enter the eating state, the relative parameters must satisfy the following eating condition.

Condition 1. (*Enter-Eating*)

$$i.h \wedge (\forall j :: Pr(i,j) = i \vee Pr(i,j) = 0)$$

Theorem 1. (*Safety*) Neighbors never eat simultaneously.

$$\neg(i.e \wedge j.e \wedge E(i,j) = 1) \qquad \textbf{\textit{Invariant 1}}$$

Proof. We assume philosopher i and philosopher j share a resource, so they are neighbors. If philosopher i satisfies Condition 1, therefore philosopher i can eat. Because $Pr(i,j) = Pr(j,i) = i$, so philosopher j doesn't satisfy Condition 1, hence, philosopher j can't eat.

The direction from i to j doesn't change until i eats, or, equivalently, *a philosopher yields its priority to all its neighbors only when it eats*, which satisfy the following rule.

Transformation Rule

$$i.e \wedge E(i,j) = 1 \Rightarrow Pr(i,j) = j \wedge Pr(j,i) = j$$

Deadlock should satisfy the next Condition.

Condition 2 (*Deadlock*). There is a cycle in graph H_i, where $i \geq 0$.

Theorem 2 (*Free-Deadlock*). If the initial state of DDPP H_0 is acyclic, there is no deadlock in the following states of DDPP, because the next state of DDPP H_i $(i > 0)$ remains acyclic.

H_i $(i \geq 0)$ *is acyclic.* ***Invariant 2***

Proof. The initial graph H_0 is acyclic, the transformation rule ensures the acyclicity of H_i $(i > 0)$, because of a eating philosopher with all of its edges are directed towards to itself, therefore, there is no circle in graph H_i $(i > 0)$.

Theorem 3 (*Free-Starvation/Progress*). Every hungry philosopher eventually eats.

Proof. A philosopher yields its priority to all its neighbors only when it eats (Transformation Rule). Consequently, an eaten philosopher doesn't eat once again until that all its neighbors have finished last-time-eating. Hence, it eats again when it acquires all the requested resources, just after the other neighbors yield priority to this eaten philosopher.

Theorem 4 (*Fairness*). Every philosopher has the same opportunity to eat. The ratio of eating for philosopher i and philosopher j is $1 : 1$.

Proof. The fairness is can be deduced from Theorem 3.

4 Categorial Semantics of Distributed Dining Philosophers Problem

Based upon the above solution, we originally present the formal semantics of DDPP using category theory in this section.

4.1 Signature

There are three phases for every philosopher. The transformations of phases also cover three actions: from thinking to hungry, from hungry to eating, and from eating to thinking. The signature of philosopher is defined according to its phases and transformations.

Definition 6. ***Signature of Philosopher***

The philosopher signature is duple $\gamma = \{\Delta, \Theta\}$, where:

- Δ is a set of all the phases of philosophers, $\Delta = \{t, h, e\}$;
- Θ is a set of action, each action represents a transformation between two phases, $\Theta = \{i.e \mapsto i.t, i.t \mapsto i.h, i.h \mapsto i.e\}$.

According to the undirected graph G given in the Sect. 3, we could define the signature category of DDPP.

Definition 7. ***Signature of DDPP***

The DDPP's signature is duple $\Sigma = \{\Upsilon, \Omega\}$, where:

- Υ is a set of all the nodes in G, each node represents a philosopher γ;
- Ω is a set of all edges in G, each edge represents the neighbor relationship between two different philosophers.

4.2 Categorial Semantics of Philosopher

The phase category $\mathbb{P}$ could be defined based on the signature of philosopher and Condition 1(in Sect. 3).

Definition 8. ***Category: Phases of Philospher*** $\mathbb{P} = (\Delta, \Gamma)$
The phase category $\mathbb{P}$ (see Fig. 2) includes a set Δ and a collection of three guarded morphisms and three identity morphisms $\Gamma = \{f_1, f_2, f_3, id_t, id_e, id_h\}$. For every object i in Δ, there is an *identity* morphism $id_i : i \to i$, which means the current phase of philosopher preserve unchanged. f_1, f_2, and f_3 separately represent three actions in Θ of philosopher's signature. These guarded morphisms are described by using the following form: $(\varsigma \rightsquigarrow f : x \to y)$, where ς is the condition of morphism $f : x \to y$. Therefore, f_1, f_2, f_3 is defined as follows.

(1) $Finished \rightsquigarrow f_1 : e \to t$ (*Finished* is a signal on the completion of eating.)
(2) $Request \rightsquigarrow f_2 : t \to h$ (*Request* is a signal denotes asking for resources.)
(3) $Condition\ 1 \rightsquigarrow f_3 : h \to e$

• Given two morphisms $f_x : i \to j$ and $f_y : j \to k$ such that $dom(f_y) = cod(f_x)$. The *composition* of f_x and f_y, has three different kinds, which are written as $f_2 \circ f_1 : e \to h$; $f_3 \circ f_2 : t \to e$ and $f_1 \circ f_3 : h \to t$;

• Composition is *associative*, that is: given $f_1 : e \to t$, $f_2 : t \to h$, and $f_3 : h \to e$, then $f_3 \circ (f_2 \circ f_1) = (f_3 \circ f_2) \circ f_1$;

• Each *identity* morphism $id_i : i \to i$ satisfies $id_i \circ f_* = f_*$ for every $f_* : j \to i$, and $f'_* \circ id_i = f'_*$ for every $f'_* : i \to j$.

Our solution does not deny the possibility of simultaneous eating for non-neighbors in terms of Condition 1 and Theorem 1.

Definition 9. ***Parallel Philosophers*** $x \parallel y$
$\mathbb{P}_k(\Gamma) = \{f_1, f_2, f_3\}$ represents a set of three transformed morphisms between two phases of philosopher k, where f_1, f_2, and f_3 is defined in Definition 8.

Given the precondition $f_i \in \mathbb{P}_x(\Gamma) \wedge g_j \in \mathbb{P}_y(\Gamma)$ where $(1 \le i, j \le 3)$, the parallel operation $f_i \parallel g_j$ means the action f_i and g_j can simultaneously take place. Hence, the parallel relationship between philosopher x and philosopher y can be divided into two cases.

Case 1: If philosopher x and philosopher y do not have common resource $(E(x, y) = 0)$, all three actions/morphisms f_i $(1 \le i \le 3)$ and g_j $(1 \le j \le 3)$ can synchronously happen at the some time.

$x \parallel y = (\forall i, j : 1 \le i \le 3 \wedge 1 \le j \le 3 : f_i \parallel g_j))$

Case 2: If philosopher x and philosopher y have common resource $(E(x, y) = 1)$, except the 3^{rd} action/morphism $f_3 : h \to e$ and $g_3 : h \to e$, the other situations can synchronously happen at the some time.

$x \parallel y = ((\forall i, j : 1 \le i \le 3 \wedge 1 \le j \le 2 : f_i \parallel g_j) \wedge (\forall i, j : 1 \le i \le 2 \wedge 1 \le j \le 3 : f_i \parallel g_j) \wedge ((f_3 \circ g_3) \vee (g_3 \circ f_3)))$

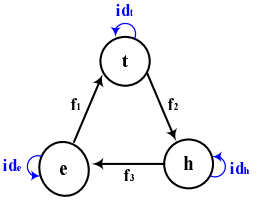


Fig. 2. Graph of Category $\mathbb{P}$

where $(f_3 \circ g_3)$ indicates that philosopher y eats before philosopher x eats if $Pr(x, y) = y$; and vice versa.

4.3 Categorial Semantics of State

A state of DDPP shows the priority of philosophers at any time, and the state changes only as some philosophers enter into the eating phase. The state category $\mathbb{C}$ could be defined based on precedence graph H.

Definition 10. ***Category: State of DDPP*** $\mathbb{C} = (\Upsilon, \Phi)$
The state category $\mathbb{C}$ is composed of Υ in Definition 7, each object represents a philosopher γ; and a morphism collection Φ, where for every morphism $f : i \to j \in \Phi$ $(i, j \in \Upsilon \wedge i \neq j)$ means that philosopher i has priority over j, and for every object i in Υ,there is an *identity* morphism $id_i : i \to i$, which indicates that the priority of i is equivalent to the priority of i.

- Given two morphisms $f : i \to j$ and $g : j \to k$ such that $dom(g) = cod(f)$. The *composition* of f and g, written $g \circ f : i \to k$, is defined and has domain $dom(f)$ and codomain $cod(g)$. The composition means the priorities of philosophers are *transitive*;
- Composition is *associative*, that is, given $f : i \to j$, $g : j \to k$, and $h : k \to w$, then $h \circ (g \circ f) = (h \circ g) \circ f$;
- Each *identity* morphism $id_i : i \to i$ satisfies $id_i \circ g = g$ for every $g : j \to i$, and $f \circ id_i = f$ for every $f : i \to j$.

Category theory supports the diagrammatic representation which visualizes the relationships between concepts. It is possible to use diagrams to express and reason about properties in a formal way.

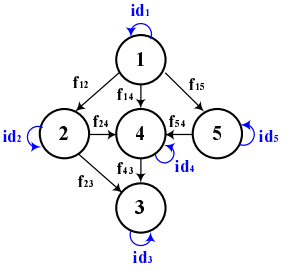


Fig. 3. Graph of Category $\mathbb{C}_{H_0}$

Example: The graph of a category $\mathbb{C}_{H_0}$ is given in Fig. 3. The diagram of a category $\mathbb{C}_{H_0}$ and its graph H_0 is a graph homomorphism $\delta : \mathbb{C}_{H_0} \to H_0$.

4.4 Categorial Semantics of States-Transition

The initial state H_0 can be depicted by a category $\mathbb{C}_{H_0}$. Now, let's continue to analyze the possible following states with an ***assumption*** that all the philosophers are hungry and want to require their resources. Obviously, the category

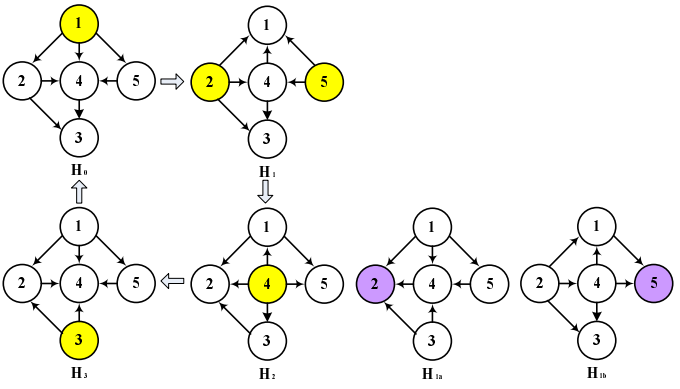


Fig. 4. States-Transition of DDPP (H_1a and H_1b are two other states after H_0)

$\mathbb{C}_{H_0}$ is only changed as soon as the 1^{st} philosopher begin eating. As demonstrated in the Fig. 4, the next state is H_1. Then, the 2^{nd} philosopher and the 5^{th} philosopher will eat simultaneously because they are not neighbors. After acquiring the four resources, the 4^{th} philosopher can eat in state H_2, and result in state H_3. Finally, the state H_3 will be return to state H_0.

The two other different states (see Fig. 4) after H_1 take place after state H_0:
Case 1: If the 2^{nd} philosopher is hungry, and the 5^{th} philosopher is thinking, the next state is H_{1a}.
Case 2: If the 2^{nd} philosopher is thinking, and the 5^{th} philosopher is hungry, the next state is H_{1b}.

Therefore, the states-transition can be formally defined as a category $\mathbb{T}$.

Definition 11. ***Category: States-Transition of DDPP*** $\mathbb{T} = (\mathfrak{C}, \Psi)$
A category DDPP-Change is composed of two collections:

- $\mathfrak{C}$ is the collection of all state categories $\mathbb{C}_i$ in DDPP, each object represents a state category of DDPP;
- Ψ is the collection of morphisms, each morphism $\psi : \mathbb{C}_i \to \mathbb{C}_j (i \neq j)$ represents the convert from a state $\mathbb{C}_i$ to another state $\mathbb{C}_j$ after some philosophers enter into eating, and for every object $\mathbb{C}_i$ in $\mathfrak{C}$, there is an *identity* morphism $id_{\mathbb{C}_i} : \mathbb{C}_i \to \mathbb{C}_i$, which means that the current state $\mathbb{C}_i$ of DDPP preserve unchanged.

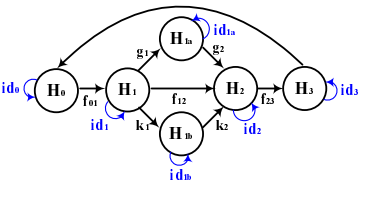


Fig. 5. Graph of Category $\mathbb{T}$

It's easy to verify the correctness of composition law, associative law and identity law in above category $\mathbb{T} = (\mathfrak{C}, \Psi)$. The graph of category $\mathbb{T}$ is illustrated in the Fig. 5, where states collection $\mathfrak{C} = \{H_0, H_1, H_{1a}, H_{1b}, H_2, H_3\}$; and morphisms collection $\Psi = \{f_{01}, f_{12}, f_{23}, f_{30}, g_1, g_2, k_1, k_2, id_0, id_1, id_2, id_3, id_{1a}, id_{1b}\}$.

Theorem 5. The diagram of category States-Transition $\mathbb{T}$ is commutative.

The commutative property of a diagram helps to establish a set of equalities between morphisms. Hence, diagrams and commutativity provide us with the ability of doing equational reasoning in a visual form.

Example: In Fig. 5, there are three different paths from H_1 to H_2. We can get the equation:

$$g_2 \circ g_1 = f_{12} = k_2 \circ k_1$$

4.5 Categorial Semantics of System

The system of DDPP consists of some process: philosopher's processes Ph_i $(1 \leq i \leq n)$ and environmental process En. Each process Ph_i has its own private signature $\gamma = (\Delta, \Theta)$. The matrixes E and Pr, which belong to the process En, can be accessed by all process Ph_i. Category theory provides the level of mathematical abstraction to describe software architectures, we define the category of processes' composition $\mathbb{S}$.

Definition 12. ***Category: Composition of Processes*** $\mathbb{S} = (\Xi, \bigoplus)$

A category $\mathbb{S}$ is composed of two collections:

- Ξ is the collection of all processes $\Xi = \{Ph_1, Ph_2, ..., Ph_n, En\}$ in DDPP, each object represents a process of DDPP;
- $\bigoplus$ is the collection of morphisms between two different processes. The *access morphism* $(\mu : En \to Ph_i)$ means that the matrix E can be read from process En to process Ph_i, and matrix Pr can be read and rewritten by process Ph_i. The *compose morphism* $(\nu : Ph_i \to (Ph_i \frown Ph_*))$ or $(\nu : Ph_i \to (Ph_* \frown Ph_i))$ indicates that process Ph_i is part of the compositive components $(Ph_i \frown Ph_*)$ or $(Ph_* \frown Ph_i)$.
- For every object P_i in Ξ there is an *identity* morphism $id_{P_i} : P_i \to P_i$, which *means that the process is not combined with other processes*, satisfying $id_{P_i} \circ g = g$ for every $g : P_j \to P_i$ and $f \circ id_{P_i} = f$ for every $f : P_i \to P_j$.

It's easier to proof the correctness of composition law and associative law. One of the characteristics of universal constructions (like pushout and colimit) is the

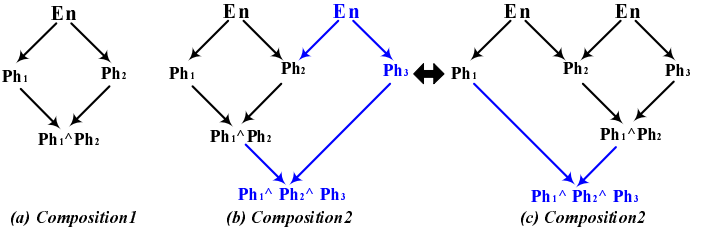


Fig. 6. Composition of Processes

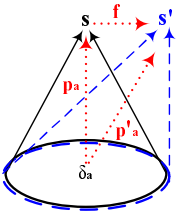


Fig. 7. Colimit of Diagram δ in category $\mathbb{S}$

ability to capture the collective behaviors of systems of interconnected components. Two processes Ph_1 and Ph_2 can be integrated into combined component $Ph_1 \frown Ph_2$ by using ***pushout*** operation (See the picture (a) in Fig. 6). As the pictures (b) and (c) elaborated, two different approaches about composing three processes Ph_1, Ph_2 and Ph_3 could be expressed by employing the two pushout operations, such that $Composition2$ is equivalent to $Composition3$.

In DDPP, we can get a entire-system after using the ***pushout*** operation for $n-1$ times. Different systems could be gained through different orders of composition. Each system will be considered as a colimit of the diagram of the category $\mathbb{S}$.

Definition 13. ***Colimit of System***
Let $\delta : I \rightarrow graph(\mathbb{S})$ be a diagram in category $\mathbb{S}$. A colimit of δ is a commutative cocone $p : \delta \rightarrow s$, where s is one of the composition of all processes, such that, for every other entire-system s' and its commutative cocone $p' : \delta' \rightarrow s'$, there is a unique morphism $f : s \rightarrow s'$ such that $f \circ p = p'$ (see the Fig. 7).

Theorem 6. Every entire-system s_i and its commutative cocone $p : \delta \rightarrow s_i$ is a colimit of $\delta : I \rightarrow graph(\mathbb{S})$.

5 Conclusions and Future Works

In this paper, we have shown how concepts and semantics of a particular model about DDPP can be formalized in a categorial framework. The purpose is to emphasize the strengths of category theory – its simple theory, diagrammatical representation and its expressive power in representing concepts of computer science. In contrast to other formal models of dining philosophers problem, the perceived benefits of our categorial formalization are as follows:

Firstly, category theory is advocated as a good mathematical structure for formalize different relationships and interactions precisely because the morphisms can be regarded as structure-preserving mappings. In our paper, we define four categories from four different levels: phases of philosophers, priority of philosophers, state-transitions and system's integration. Each category has its own objects and morphisms, which provide an formal way to express the relationships between objects.

Secondly, a distinctive attribute of category theory as a mathematical formalization is that it is essentially graphical [26]. The diagrammatical nature

of category theory certainly not only help to express and reason about some properties of DDPP in a formal way, but also enhance the traceability and understandability the solution of DDPP.

Additionally, category theory offers techniques for manipulating and reasoning about system configuration represented as the colimit of a diagrams, which could express how a system is configured in term of several simpler components (philosopher's processes and environmental process) and interconnections between them. Hence, we can get a model of the global behaviors of the system and the morphisms that relate the processes to the global system.

We will continue our research on the other distributed resource allocation problems (such as committee coordination problem [27] and dynamic resources allocation problem [33]), which will deeply explain the superiority, efficiency and dependability of categorial formalization.

Acknowledgments. The authors thanks Professor José Luiz Fiadeiro for detailed comments and discussion about this work.

References

1. Dijkstra, E.W.: Hierarchical Ordering of Sequenial Processes. In: Operating Systems Techniques. Academic Press, London (1971)
2. Dijkstra, E.W.: Two Starvation-free Solutions of a General Exclusion Problem. EWD 625, Platanstraat 5, AI Nuenen, The Netherlands (1978)
3. Awerbuch, B., Saks, M.: A dining philosophers Algorithm With Polynomial Response Time. In: Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, pp. 65–74 (1990)
4. Page, I.A., Jacob, R.T., Chern, S.E.: Optimal Algorithms for Distributed Resource Allocation. IEEE Distributed Computing (1991)
5. Lehmann, D., Rabin, M.O.: On The Advantages of Free Choice: a Symetric and Fully Distributed Solution for The Dining Philosophers Problem. In: Roscoe, A.W. (ed.) A classical mind: essays in honour of C.A.R. Hoare, ch. 20, pp. 333–352. Prentice Hall, Englewood Cliffs (1994)
6. Duflot, M., Fribourg, L., Picaronny, C.: Randomized Dining Philosophers Without Fairness Assumption. Distrib. Comput. 17(1), 65–76 (2004)
7. Garg, V.K.: Analysis of Distributed Systems With Many Identical Processes. Distributed Computing Systems, 358–365 (1988)
8. Aggarwal, S., Barbara, D., Meth, K.Z.: A Software Environment for The Specification And Analysis of Problems of Coordination And Concurrency. IEEE Transactions on Software Engineering 14(3), 280–290 (1988)
9. Parashkevov, A.N., Yantchev, J.: ARC-a Tool for Efficient Refinement And Equivalence Checking for CSP. In: ICAPP 1996, pp. 68–75 (1996)
10. Furia, C.A., Rossi, M., Dino, M., Morzenti, A.: Automated Compositional Proofs for Real-time Systems. Theoretical Computer Science (2007)
11. Chandy, K.M., Misra, J.: The Drinking Philosophers Problem. Prog. Lang. Syst. 6(4), 632–646 (1984)
12. Rine, D.C.: A Category Theory For Programming Languages. Mathematical System Theory 7(4), 304–317 (1973)

13. Reynolds, J.C.: Using Category Theory to Design Programming Languages. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 62–63. Springer, Heidelberg (2009)
14. Fiadeiro, J.L., Maibaum, T.: Categorical Semantics of Parallel Program Design. Science of Computer Programming 28(2-3), 111–138 (1997)
15. Minamizono, K., Katai, O., Shiose, T., Kawakami, H.: Analyses of Design Processes Based on Category Theory and Channel Theory. In: SICE Annual Conference in Fukui (2003)
16. Buisse, A., Dybjer, P.: The Interpretation of Intuitionistic Type Theory in Lacolly Cartesian Closed Categories an Intuitionistic Perspective. Electronic Notes in Theoretical Computer Science 218, 21–23 (2008)
17. Buisse, A., Dybjeri, P.: Towards Formallizing Categorical Models of Type Theory in Type Theory. Electronic Notes in Theoretical Computer Science 196, 137–151 (2008)
18. Ehrig, H., Grobe-Rhode, M., Wolter, U.: Applications of Category Theory to the Area of Algebraic Specification in Computer Science. Applied Categorical Structures 6, 1–35 (1998)
19. Altuncher, J.A., Panangaden, P.: A Mechanically Assisted Constructive Proof in Category Theory. In: Stickel, M.E. (ed.) CADE 1990. LNCS, vol. 449, pp. 673–674. Springer, Heidelberg (1990)
20. Kozen, D., Kreitz, C., Richter, E.: Automating Proofs in Category Theory. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 392–407. Springer, Heidelberg (2006)
21. Fiadeiro, J.L., Maibaum, T.: A Mathematical Toolbox for Software Architect. In: Proc. 8th Int. Workshop on Software Sepcification and Design, pp. 44–55. IEEE Computer Science Press, Silver Springer, MD (1996)
22. Yang, X., Hou, J., Wan, J.: Formal Semantic Meanings of Architecture-Centric Model Mapping. In: Xu, M., Zhan, Y.-W., Cao, J., Liu, Y. (eds.) APPT 2007. LNCS, vol. 4847, pp. 640–649. Springer, Heidelberg (2007)
23. Crole, R.L.: Basic Category Theory for Models of Syntax. In: course notes for Summer School on Generic Programming, SSGP's (2002)
24. Lenisa, M., Power, J., Watanabe, H.: Category Theory for Operational Semantics. Theoretical Computer Science 327, 135–154 (2004)
25. van Oosten, J.: Basic Category Theory (2002)
26. Fiadeiro, J.L.: Categories for Software Engnieering. Springer, Heidelberg (2005)
27. Chandy, K.M., Misra, J.: Parallel Program Design: A Foundation. Addison-Wesley, Reading (1988)
28. Milner, R.: Communiciation and Concurrency. Prentice Hall, Englewood Cliffs (1989)
29. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Englewood Cliffs (1985)
30. Moszkowshi, B.: Executing Temporal Logic Porgrams. Cambridge University Press, Cambridge (1986)
31. Duan, Z., Tian, C., Zhang, L.: A decision procedure for propositional projection temporal logic with infinite models. Acta Informatica 45(1) (2008)
32. Dijkstra, E.W.: Cooperating Sequential Proccess. In: Genuys, F. (ed.) Programming Langugages. Academic Press, London (1968)
33. Weidman, E.B., Page, I.P., Pervin, W.J.: Explicit Dynamic Exclusion Algorithm. In: Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, pp. 7142–7149. IEEE, Los Alamitos (1991)

Approximation Algorithms for the Capacitated Domination Problem*

Mong-Jen Kao and Han-Lin Chen

Department of Computer Science and Information Engineering,
National Taiwan University, Taiwan
d97021@csie.ntu.edu.tw, kalent37@ms89.url.com.tw

Abstract. We consider the *Capacitated Domination* problem, which models a service-requirement assignment scenario and is a generalization to the well-known *Dominating Set* problem. In this problem, given a graph with three parameters defined on each vertex, namely cost, capacity, and demand, we want to find an assignment of demands to vertices of least cost such that the demand of each vertex is satisfied subject to the capacity constraint of each vertex providing the service.

In terms of polynomial time approximations, we present logarithmic approximation algorithms with respect to different demand assignment models on general graphs. On the other hand, from the perspective of parameterization, we prove that this problem is *W[1]*-hard when parameterized by a structure of the graph called treewidth. Based on this hardness result, we present exact fixed-parameter tractable algorithms with respect to treewidth and maximum capacity of the vertices. This algorithm is further extended to obtain pseudo-polynomial time approximation schemes for planar graphs.

1 Introduction

For decades, *Dominating Set* problem has been one of the most fundamental and well-known problems in both graph theory and combinatorial optimization. Given a graph $G = (V, E)$ and an integer k, *Dominating Set* asks for a subset $D \subseteq V$ whose cardinality does not exceed k such that every vertex in the graph either belongs to this set or has a neighbor which does. As this problem is known to be NP-hard, approximation algorithms have been proposed in the literature. On one hand, a simple greedy algorithm is shown to achieve a guaranteed ratio of $O(\ln n)$ [5,17,22], where n is the number of vertices, which is later proven to be the approximation threshold by Feige [9]. On the other hand, algorithms based on dual-fitting provide a guaranteed ratio of Δ [15], where Δ is the maximum degree of the vertices of the graph.

In addition to polynomial time approximations, *Dominating Set* has its special place from the perspective of parameterized complexity as well [8,10,23]. In

* This work was supported in part by the National Science Council, Taipei 10622, Taiwan, under the Grants NSC98-2221-E-001-007-MY3, NSC98-2221-E-001-008-MY3, NSC97-2219-E-001-001, and NSC97-2745-P-001-001.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 185–196, 2010.

contrast to *Vertex Cover*, which is fixed-parameter tractable (FPT), *Dominating Set* has been proven to be *W[2]*-complete when parameterized by solution size, in the sense that no fixed-parameter algorithm exists (with respect to solution size) unless FPT= *W[2]*. Though *Dominating Set* is a fundamentally hard problem in the parameterized W-hierarchy, it has been used as a benchmark problem for developing *sub-exponential time* parameterized algorithms [1,6,11] and *linear size kernels* have been obtained in planar graphs [2,10,12,23], and more generally, in graphs that exclude a fixed graph H as a minor.

Besides *Dominating Set* problem itself, a vast body of work has been proposed in the literature, considering possible variations from purely theoretical aspects to practical applications. In particular, variations of *Dominating Set* problem, ranging from strategic decisions, such as locating radar stations or emergency services, to computational biology and to voting systems, occur in numerous practical settings. See [14,24] for a detailed survey. For example, Haynes et al. [13] considered *Power Domination Problem* in electric networks [13,21] while Wan et al. [25] considered *Connected Domination Problem* in wireless ad hoc networks.

Motivated by a general service-requirement assignment model, Kao et al., [19] considered a generalized domination problem called *Capacitated Domination*. In this problem, the input graph is given with tri-weighted vertices, referred to as *cost*, *capacity*, and *demand*, respectively. The demand of a vertex stands for the amount of service it requires from its adjacent vertices (including itself) while the capacity of a vertex represents the amount of service it can provide when it's selected as a server. The goal of this problem is to find a dominating multi-set as well as a demand assignment function such that the overall cost of the multi-set is minimized. For different underlying applications, there are two different demand assignment models, namely *splittable* demand model and *unsplittable* demand model, depending on whether or not the demand of a vertex is allowed to be served by different vertices. Moreover, depending whether the number of copies, or *multiplicity* of each vertex in the dominating multi-set, is limited, there are different models, which we referred to as *hard* capacity and as *soft* capacity.

Kao et al., [19] considered the problem with soft capacity and splittable demand model and provided a $(\Delta + 1)$-approximation for general graphs, where Δ is the degree of the graph. They proved that the problem remains NP-hard even when the input graph is restricted to a tree, for which they also presented a polynomial time approximation scheme. Dom et al., [7] considered the hard capacitated domination problem with uniform demand and showed that this problem is *W[1]*-hard even when parameterized by treewidth and solution size.

Our contribution. In this paper, we consider the (soft) *Capacitated Domination* problem and present logarithmic approximation algorithms with respect to different demand assignment models on general graphs. Specifically, we provide a $(\ln n)$-approximation for weighted unsplittable demand model, a $(4 \ln n + 2)$-approximation for weighted splittable demand model, and a $(2 \ln n + 1)$-approximation for unweighted splittable demand model, where n is the number of vertices. Together with the $(\Delta + 1)$-approximation result given by Kao et al.,

[19], this establishes a corresponding near-optimal approximation result to the original *Dominating Set* problem and closes the problem of generally approximating this problem. Although the result may look natural, the greedy choice we make is not obvious when non-uniform capacity as well as non-uniform demand is taken into consideration. On the other hand, from the perspective of parameterization, we prove that this problem is *W[1]*-hard when parameterized by a structure of the graph, called the treewidth, and present exact FPT algorithms with respect to both treewidth and maximum capacity of the vertices. This algorithm is further extended to obtain pseudo-polynomial time approximation schemes for planar graphs, based on a framework due to Baker [3].

The rest of this paper is organized as follows. In Section 2, we give formal definitions and notation adopted in the paper. In Section 3, we present our ideas and algorithms that achieve the aforementioned approximation guarantees. We present the parameterized results in Section 4 and conclude by listing some future work in Section 5. Note that most proofs as well as detailed algorithm pseudo-codes are omitted due to the space limit and can be found in the full version of this work [18].

2 Preliminary

We assume that all the graphs considered in this paper are simple and undirected. Let $G = (V, E)$ be a graph. The set of neighbors of a vertex $v \in V$ is denoted by $N_G(v) = \{u : (u, v) \in E\}$. The closed neighborhood of $v \in V$ is denoted by $N_G[v] = N_G(v) \cup \{v\}$. The subscript G in $N_G[v]$ will be omitted when there is no confusion.

Consider a graph $G = (V, E)$ with tri-weighted vertices, referred to as the cost, the capacity, and the demand of each vertex $u \in V$, denoted by $w(u)$, $c(u)$, and $d(u)$, respectively. Let D denote a multi-set of vertices of V and for any vertex $u \in V$, let $x_D(u)$ denote the *multiplicity* of u or the number of times of u in D. The cost of D, denoted $w(D)$, is defined to be $w(D) = \sum_{u \in D} w(u) \cdot x_D(u)$.

Definition 1 (Capacitated Dominating Set). *A vertex multi-subset D is said to be a feasible capacitated dominating set with respect to a demand assignment function f if the following conditions hold.*

- **Demand constraint:** $\sum_{u \in N_G[v]} f(v, u) \geq d(v)$, *for each* $v \in V$.
- **Capacity constraint:** $\sum_{u \in N_G[v]} f(u, v) \leq c(v) \cdot x_D(v)$, *for each* $v \in V$.

Given a problem instance, the capacitated domination problem asks for a capacitated dominating multi-set D and demand assignment function f such that $w(D)$ is minimized. For unsplittable demand model we require that $f(u, v)$ is either 0 or $d(u)$ for each edge $(u, v) \in E$. Note that since it is already NP-hard[1] to compute a feasible demand assignment function from a given feasible capacitated dominating multi-set when the demand cannot be split, it is natural

[1] This can be verified by making a reduction from Subset Sum.

to require the demand assignment function to be specified, in addition to the optimal vertex multi-set itself.

Parameterized complexity is a well-developed framework for studying the computationally hard problem [8,10,23]. A problem is called *fixed-parameter tractable* (FPT) with respect to a parameter k if it can be solved in time $f(k)\cdot n^{O(1)}$, where f is a computable function depending only on k. Problems (along with its defining parameters) belonging to *W[t]*-hard for any $t \geq 1$ are believed not to admit any FPT algorithms (with respect to the specified parameters). Now we define the notion of parameterized reduction.

Definition 2. *Let A and B be two parameterized problems. We say that A reduces to B by a standard parameterized reduction if there exists an algorithm Φ that transforms (x,k) into $(x',g(k))$ in time $f(k)\cdot|x|^{\alpha}$, where $f,g:\mathcal{N}\rightarrow\mathcal{N}$ are arbitrary functions and α is a constant independent of $|x|$ and k, such that $(x,k)\in A$ if and only if $(x',g(k))\in B$.*

Next we define the concept of *tree decomposition* [4,20].

Definition 3 (Tree Decomposition of a Graph). *A tree decomposition of a graph $G=(V,E)$ is a pair $(X=\{X_i : i\in I\}, T=(I,F))$ where each node $i\in I$ has associated with it a subset of vertices $X_i\subseteq V$, called the bag of i, such that*

1. *Each vertex belongs to at least one bag: $\bigcup_{i\in I} X_i = V$.*
2. *For all edges, there is a bag containing both its end-points.*
3. *For each $v\in V$, the set of nodes $\{i\in I : v\in X_i\}$ induces a subtree of T.*

The width of a tree decomposition is $\max_{i\in I}|X_i|$. The treewidth of a graph G is the minimum width over all tree decompositions of G.

3 Logarithmic Approximation

In this section, we present logarithmic approximation algorithms for capacitated domination problems with respect to different cost and demand models. Specifically, we provide a $(\ln n)$-approximation for weighted unsplittable demand model, a $(4\ln n+2)$-approximation for weighted splittable demand model, and a $(2\ln n+1)$-approximation for unweighted splittable demand model, where n is the number of vertices.

The main idea is based on greedy approach in the sense that we keep choosing a vertex with the best efficiency in each iteration until the whole graph is dominated. By best efficiency we mean the maximum cost-efficiency ratio defined for each vertex in the remaining graph. We describe the results in more detail in the following subsections.

3.1 Weighted Unsplittable Demand

Let U be the set of vertices which are not dominated yet. Initially, we have $U=V$. For each vertex $u\in V$, let $N_{ud}[u]=U\cap N[u]$ be the set of undominated

vertices in the closed neighborhood of u. Without loss of generality, we shall assume that the elements of $N_{ud}[u]$, denoted by $v_{u,1}, v_{u,2}, \ldots, v_{u,|N_{ud}[u]|}$, are sorted in non-decreasing order of their demands in the remaining section.

In each iteration, the algorithm chooses a vertex of the most efficiency from V, where the efficiency of a vertex, say u, is defined by the largest effective-cost ratio of the number of vertices dominated by u over the total cost. That is, $\max_{1 \le i \le |N_{ud}[u]|} i/\left(w(u) \cdot x_u(i)\right)$, where $x_u(i) = \left\lceil \sum_{1 \le j \le i} d(v_{u,j})/c(u) \right\rceil$ is the number of copies of u selected in order to dominate $v_{u,1}, v_{u,2}, \ldots$, and $v_{u,i}$.

In iteration j, let OPT_j be the cost of the optimal solution for the remaining problem instance, which is clearly upper bounded by the cost, OPT, of the optimal solution for the input instance. Let the number of undominated vertices at the beginning of iteration j be n_j, and the number of vertices that are newly dominated in iteration j be k_j.

Denote by S_j the cost in iteration j. Note that $S_j = w(u) \cdot x_u(k_j)$, where u is the most efficient vertex chosen in iteration j. Assume that the algorithm repeats for m iterations.

Lemma 1. *For each j, $1 \le j \le m$, we have $S_j \le \frac{k_j}{n_j} \cdot OPT_j$.*

Theorem 1. *A $(\ln n)$-approximation for weighted capacitated domination problem with unsplittable demands can be computed in polynomial time, where n is the number of vertices.*

3.2 Weighted Splittable Demand

In this section, we present an algorithm that produces a $(4 \ln n + 2)$-approximation for the *weighted capacitated domination problem with splittable demand*. The difference between this algorithm and the previous one lies in the way we handle the demand assignment. In each iteration the demand of a vertex may be partially served. The unsatisfied portion of the demand is called *residue demand*. For each vertex $u \in V$, let $rd(u)$ be the residue demand of u. $rd(u)$ is set equal to $d(u)$ initially, and will be updated accordingly when a portion of the residue demand is assigned. u is said to be completely satisfied when $rd(u) = 0$.

We will inherit the notation used in the previous section. We assume that the elements of $N_{ud}[u]$, written as $v_{u,1}, v_{u,2}, \ldots, v_{u,|N_{ud}[u]|}$, are sorted according to their demands in non-decreasing order.

In each iteration, the algorithm performs two greedy choices. First, the algorithm chooses the vertex of the most efficiency from V, where the efficiency is defined similarly as in the previous section with some modification since the demand is splittable.

For each vertex $u \in V$, let j_u with $0 \le j_u \le |N_{ud}[u]|$ be the maximum index such that $c(u) \ge \sum_{i=1}^{j_u} rd(v_{u,i})$. Let $X(u) = \sum_{i=1}^{j_u} rd(v_{u,i})/d(v_{u,i})$ be the sum of the effectiveness over the vertices whose residue demand could be completely served by a single copy of u. In addition, we let $Y(u) = (c(u) - \sum_{i=1}^{j_u} rd(v_{u,i}))/d(v_{u,j_u+1})$ if $j_u < |N_{ud}[u]|$ and $Y(u) = 0$ otherwise. The efficiency of u is defined as $(X(u) + Y(u))/w(u)$.

ALGORITHM *Split-Log-Approx*

1: $rd(u) \longleftarrow d(u)$, and $map(u) \longleftarrow \phi$ for each $u \in V$.
2: **while** there exist vertices with non-zero residue demand **do**
3: // 1^{st} **greedy choice**
4: Pick a vertex in V with the most efficiency, say u.
5: **if** j_u equals 0 **then**
6: Assign this amount $c(u) \cdot \left\lfloor \frac{rd(v_{u,1})}{c(u)} \right\rfloor$ of residue demand of $v_{u,1}$ to u.
7: $map(v_{u,1}) \longleftarrow \{u\}$
8: **else**
9: Assign the residue demands of the vertices in $\{v_{u,1}, v_{u,2}, \ldots, v_{u,j_u}\}$ to u.
10: **if** $j_u < |N_{ud}[u]|$ **then**
11: Assign this amount $c(u) - \sum_{i=1}^{j_u} rd(v_{u,i})$ of residue demand of v_{u,j_u+1} to u.
12: $map(v_{u,j_u+1}) \longleftarrow map(v_{u,j_u+1}) \cup \{u\}$
13: **end if**
14: **end if**
15:
16: // 2^{nd} **greedy choice**
17: **if** there is a vertex u with $0 < rd(u) < \frac{1}{2} \cdot d(u)$ **then**
18: Satisfy u by doubling the demand assignment of u to vertices in $map(u)$.
19: **end if**
20: **end while**
21: compute from the assignment the cost of the dominating set, and return the result.

Fig. 1. The pseudo-code for the weighted splittable demand model

Second, the algorithm maintains for each vertex $u \in V$ a set of vertices, denoted by $map(u)$, which consists of vertices that have partially served the demand of u before u is completely satisfied. That is, for each $v \in map(u)$ we have a non-zero demand assignment of u to v. Whenever there exists a vertex u whose residue demand is below half of its original demand, i.e., $0 < rd(u) < \frac{1}{2} \cdot d(u)$, after the first greedy choice, the algorithm immediately doubles the demand assignment of u to the vertices in $map(u)$. Note that in this way, we can completely satisfy the demand of u since $\sum_{v \in map(u)} f(u,v) > \frac{1}{2} \cdot d(u)$. A high-level description of this algorithm is presented in Figure 1.

Observation 1. *After each iteration, the residue demand of each unsatisfied vertex is at least half of its original demand.*

Clearly, the observation holds in the beginning when the demand of each vertex is not yet assigned. For later stages, whenever there exists a vertex u for which $0 < rd(u) < \frac{1}{2} \cdot d(u)$, it's always sufficient to double the demand assignment $f(u,v)$ of u to v for each $v \in map(u)$. If $map(u)$ is only modified under the condition $0 < j_v < |N_{ud}[v]|$, (line 12 in Figure 1), then $map(u)$ contains exactly the set of vertices that have partially served u. Since $rd(u) < \frac{1}{2} \cdot d(u)$, it's sufficient to double the demand assignment in this case so $d(u)$ is completely satisfied. If

$map(u)$ is reassigned through the condition $j_v = 0$ for some stage, then we have $c(v) < rd(u) \leq d(u)$. Since we assign this amount $c(v) \cdot \lfloor rd(u)/c(v) \rfloor$ of residue demand of u to v, this leaves at most half of the original residue demand and u will be satisfied by doubling this assignment.

Let the cost incurred by the first greedy choice be S_1 and the cost by the second choice be S_2. Notice that S_2 is bounded above by S_1. In the following, we will bound the cost S_1. For each iteration j, let u_j be the vertex of the maximum efficiency and OPT_j be the cost of the optimal solution for the remaining problem instance. Let $n_j = \sum_{u \in V} rd(u)/d(u)$ denote the sum of effectiveness of each vertex in the remaining problem instance at the beginning of this iteration. Let $S_{1,j}$ be the cost incurred by the first greedy choice in iteration j. Assume that the algorithm repeats for m iterations. We have the following lemma.

Lemma 2. *For each j, $1 \leq j \leq m$, we have*

- $S_{1,j} \leq \frac{n_j - n_{j+1}}{n_j} \cdot OPT_j$,
- $n_j - n_{j+1} \geq \frac{1}{2}$, *and*
- $\sum_{j=1}^{m-1} \lceil n_j - n_{j+1} \rceil \,/\, \lfloor n_j \rfloor \leq 2 \ln n$.

By Lemma 2 we have $\sum_{j=1}^{m} S_{1,j} \leq \sum_{j=1}^{m-1} \frac{n_j - n_{j+1}}{n_j} \cdot OPT_j + \frac{n_m}{n_m} \cdot OPT_m \leq \left(\sum_{j=1}^{m-1} \lceil n_j - n_{j+1} \rceil \,/\, \lfloor n_j \rfloor + 1 \right) \cdot OPT$, since $\lfloor r \rfloor \leq r \leq \lceil r \rceil$ for any real number r and $OPT_j \leq OPT$ for each $1 \leq j \leq m$.

Theorem 2. *Algorithm* Split-Log-Approx *computes a $(4 \ln n + 2)$-approximation, where n is the number of vertices, for weighted capacitated domination problem with splittable demands.*

3.3 Unweighted Splittable Demand

In this section, we consider the *unweighted capacitated domination problem with splittable demand* and present a $(2 \ln n + 1)$-approximation. In this case the weight $w(v)$ of each vertex $v \in V$ is considered to be 1 and the cost of the capacitated domination multiset D corresponds to the total multiplicity of the vertices in D. To this end, we first make a greedy reduction on the problem instance by spending at most $1 \cdot OPT$ cost such that it takes at most one copy to satisfy each remaining unsatisfied vertex. Then we show that a $(2 \ln n)$-approximation can be computed for the remaining problem instance, based on the same framework of Section 3.2.

For each $u \in V$, let g_u be the vertex in $N[u]$ with the maximum capacity. First, for each $u \in V$, we assign this amount $c(g_u) \cdot \left\lfloor \frac{d(u)}{c(g_u)} \right\rfloor$ of the demand of u to g_u. Let the cost of this assignment be S, then we have the following lemma.

Lemma 3. *We have $S \leq OPT$, where OPT is the cost of the optimal solution.*

In the following, we will assume that $d(u) \leq c(g_u)$, for each $u \in V$. The algorithm of Section 3.2 is slightly modified. In particular, for the second greedy choice, whenever $rd(u) < d(u)$ for some vertex $u \in V$, we immediately assign the residue demand of u to g_u.

Theorem 3. *A $(2\ln n + 1)$-approximation for weighted capacitated domination problem with unsplittable demands can be computed in polynomial time, where n is the number of vertices.*

4 Parameterized Results

4.1 Hardness Results

In this section we show that *Capacitated Domination Problem* is *W[1]*-hard when parameterized by treewidth by making a reduction from *k-Multicolor Clique*, a restriction of *k-Clique* problem.

Definition 4 (Multicolor Clique). *Given an integer k and a connected undirected graph $G = \left(\bigcup_{i=1}^{k} V[i], E\right)$ such that $V[i]$ induces an independent set for each i, the* Multicolor Clique *problem asks whether or not there exists a clique of size k in G.*

Given an instance (G, k) of Multicolor Clique, we will show how an instance $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of *Capacitated Domination* with treewidth $O(k^2)$ can be built such that G has a clique of size k if and only if $\mathcal{G}$ has a capacitated dominating set of cost at most $k' = (3k^2 - k)/2$. For convenience, we shall distinguish the vertices of $\mathcal{G}$ by referring to them as *nodes*.

Let N be the number of vertices. Without loss of generality, we label the vertices of G by numbers, denoted $label(v)$, $v \in V$, between 1 and N. For each $i \neq j$, let $E[i, j]$ denote the set of edges between $V[i]$ and $V[j]$. The graph $\mathcal{G}$ is defined as follows. For each i, $1 \leq i \leq k$, we create a node x_i with $w(x_i) = k' + 1$, $c(x_i) = 0$, and $d(x_i) = 1$. For each $u \in V[i]$, we have a node $\overline{u}$ with $w(\overline{u}) = 1$, $c(\overline{u}) = 1 + (k-1)N$, and $d(\overline{u}) = 0$. We also connect $\overline{u}$ to x_i. For convenience, we refer to the star rooted at x_i as vertex star T_i.

Similarly, for each $1 \leq i < j \leq k$, we create a node y_{ij} with $w(y_{ij}) = k' + 1$, $c(x_i) = 0$, and $d(x_i) = 1$. For each $e \in E[i, j]$ we have a node $\overline{e}$ with $w(\overline{e}) = 1$, $c(\overline{e}) = 1 + 2N$, and $d(\overline{e}) = 0$. We connect $\overline{e}$ to y_{ij}. We refer to the star rooted at y_{ij} as edge star T_{ij}. The selection of nodes in T_i and T_{ij} in the capacitated dominating set will correspond to the choices made in selecting the vertices that form a clique in G.

In addition, for each $i \neq j$, $1 \leq i, j \leq k$, we create two bridge nodes $b^1_{i,j}$, $b^2_{i,j}$ with $w(b^1_{i,j}) = w(b^2_{i,j}) = 1$ and $d(b^1_{i,j}) = d(b^2_{i,j}) = 1$. The capacities of the bridge nodes are to be defined later. Now we describe the way how stars T_i and T_{ij} are connected to bridge nodes such that the reduction claimed above holds. For each $i \neq j$, $1 \leq i, j \leq k$ and for each $v \in V[i]$, we create two propagation nodes $p^1_{v,i,j}$, $p^2_{v,i,j}$ and connect them to $\overline{v}$. Besides, we connect $p^1_{v,i,j}$ to $b^1_{i,j}$ and $p^2_{v,i,j}$ to $b^2_{i,j}$. We set $w(p^1_{v,i,j}) = w(p^2_{v,i,j}) = k' + 1$ and $c(p^1_{v,i,j}) = c(p^2_{v,i,j}) = 0$. The demands of $p^1_{v,i,j}$ and $p^2_{v,i,j}$ are set to be $d(p^1_{v,i,j}) = label(v)$ and $d(p^2_{v,i,j}) = N - label(v)$. For each $1 \leq i < j \leq k$ and for each $e = (u, v) \in E[i, j]$, we create four propagation nodes $p^1_{e,i,j}$, $p^2_{e,i,j}$, $p^1_{e,j,i}$, and $p^2_{e,j,i}$ with zero capacity and $k' + 1$ cost. Without

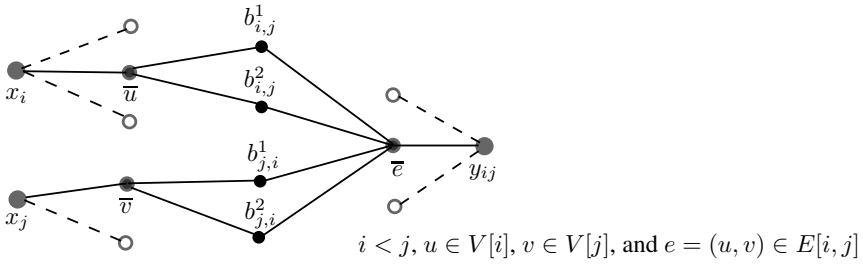


Fig. 2. The connections between stars and bridge nodes

loss of generality, we assume that $u \in V[i]$ and $v \in V[j]$. The demands of the four nodes are set as the following: $d(p^1_{e,i,j}) = N - label(u)$, $d(p^2_{e,i,j}) = label(u)$, $d(p^1_{e,j,i}) = N - label(v)$, and $d(p^2_{e,j,i}) = label(v)$. Finally, for each bridge node b, we set $c(b) = \sum_{u \in N[b]} d(b) - N$.

Lemma 4. *The treewidth of $\mathcal{G}$ is $O(k^2)$. Furthermore, G admits a clique of size k if and only if $\mathcal{G}$ admits a capacitated dominating set of cost at most $k' = (3k^2 - k)/2$.*

Theorem 4. *The* Capacitated Domination problem *is W[1]-hard when parameterized by treewidth, regardless of the demand model.*

4.2 FPT Algorithms on Graphs of Bounded Treewidth

In this section we show that *Capacitated Domination Problem* with unsplittable demand is FPT with respect to treewidth and maximum capacity by giving a $2^{2k(\log M+1)+\log k+O(1)} \cdot n$ exact algorithm. To this end, we give a dynamic programming algorithm on a so-called *nice tree decomposition* [20] of the input graph G.

Definition 5 (Nice Tree Decomposition). *A tree decomposition (X, T) is a nice tree decomposition if one can root T in such a way that each node $i \in I$ is of one of the four following types. (1)* Leaf*: node i is a leaf of T, and $|X_i| = 1$. (2)* Join*: node i has exactly two children, say j_1 and j_2, and $X_i = X_{j_1} = X_{j_2}$. (3)* Introduce*: node i has exactly one child, say j, and there is a vertex $v \in V$ such that $X_i = X_j \cup \{v\}$. (4)* Forget*: node i has exactly one child, say j, and there is a vertex $v \in V$ such that $X_j = X_i \cup \{v\}$.*

Given a tree decomposition of width k, a nice tree decomposition T of the same width can be found in linear time [20]. In the following, without loss of generality, we shall assume that the bag associated with the root of T is empty. For each node i in the tree T, let T_i be the subtree rooted at i and $Y_i := \bigcup_{j \in T_i} X_j$. Starting from the leaf nodes of T, our algorithm proceeds in a bottom-up manner and maintains for each node i of T a table A_i whose columns consist of (a) P with

$P \subseteq X_i$ indicating the set of vertices in X_i that have been served in future stages, and (b) $rc(u)$ with $0 \leq rc(u) < c(u)$ indicating the residue capacity of u, for each $u \in X_i$.

Clearly, each row of A_i corresponds to a possible configuration consisting of the unsatisfied vertices and the residue capacity of each vertex in X_i that can be used. The algorithm computes for each row of A_i the cost of the optimal solution to the subgraph induced by Y_i under the constraint that the configuration of vertices in X_i agrees with that specified by the values of the row.

In the following, we describe the computation of the table A_i for each node i in the tree T in more detail. In order to keep the content clean, we use the terms "insert a new row" and "replace an old row by the new one" interchangeably. Whenever the algorithm attempts to insert a new row into a table while another row with identical configuration already exists, the one with the smaller cost will be kept. We have the following situations.

- i **is a leaf node.** Let $X_i = \{v\}$. We add two rows to the table A_i which correspond to cases whether or not v is served.
- i **is an introduce node.** Let j be the child of i, and let $X_i = X_j \cup \{v\}$. The data in A_j is basically inherited by A_i.We extend A_i by considering, for each existing row r in A_j, all $2^{|X_j \setminus P_r|}$ possible ways of choosing vertices in $X_j \setminus P_r$ to be assigned to v. In addition, v can be either unassigned or assigned to any vertex in X_i.
- i **is a forget node.** Let j be the child of i, and let $X_i = X_j \setminus \{v\}$. In this case, for each row $r \in A_j$ such that $v \in P_r$, we insert a row r' to A_i identical to r except for the absence of v in $P_{r'}$.
- i **is a join node.** Let j_1 and j_2 be the two children of i in T. We consider every pair of rows r_1 and r_2 such that $P_{r_1} \cap P_{r_2} = \phi$, where $r_1 \in A_{j_1}$ and $r_2 \in A_{j_2}$. For each such pair of rows (r_1, r_2), we insert a new row r to A_i with $P_r = P_{r_1} \cup P_{r_2}$, $rc_r(u) = (rc_{r_1}(u) + rc_{r_2}(u)) \bmod c(u)$, for each $u \in X_i$, and $cost(r) = cost(r_1) + cost(r_2) - \sum_{u \in X_i} \left\lfloor \frac{rc_{r_1}(u) + rc_{r_2}(u)}{c(u)} \right\rfloor$.

Theorem 5. *Capacitated Domination problem with unsplittable demand on graphs of bounded treewidth can be solved in time* $2^{2k(\log M+1)+\log k+O(1)} \cdot n$*, where* k *is the treewidth and* M *is the largest capacity.*

We state without going into details that by suitably replacing the set P_i with the residue demand $rd_i(u)$ for each vertex $u \in X_i$ in the column of the table we maintained, the algorithm can be modified to handle the splittable demand model. We have the following corollary.

Corollary 1. *Capacitated Domination problem with splittable demand on graphs of bounded treewidth can be solved in time* $2^{(2M+2N+1)\log k+O(1)} \cdot n$*, where* k *is the treewidth,* M *is the largest capacity, and* N *is the largest demand.*

4.3 Extension to Planar Graphs

In this section we extend the above FPT algorithms based on a framework due to Baker [3] to obtain a pseudo-polynomial time approximation scheme

for planar graphs. In particular, for unsplittable demand model, given a planar graph G with maximum capacity M and an integer k, the algorithm computes an $(1 + \frac{4}{k-1})$-approximation in time $O(2^{2k(\log M+1)+2\log k}n)$, where n is the number of vertices. Taking $k = \lceil c \log n \rceil$, where c is some constant, we get a pseudo-polynomial time approximation algorithm which converges toward optimal as n increases. On the other hand, for splittable demand model, we have a pseudo-polynomial time approximation scheme in $O(2^{(2M+2N+1)\log k+O(1)} \cdot n)$ time, where N is the maximum demand. To get rid of the factor N, we could apply the transformation used in Section 3.3 and Lemma 3 in advance and obtain a $(2 + \frac{4}{k-1})$-approximation in $O(2^{(4M+1)\log k+O(1)} \cdot n)$ time.

This is done as follows. Given a planar graph G, we generate a planar embedding and retrieve the vertices of each level using the linear-time algorithm of Hopcroft and Tarjan [16]. Let m be the number of levels of this embedding. Let OPT be the cost of the optimal capacitated dominating set of G, and OPT_j be the cost contributed by vertices at level j.

Since $\sum_{0\le i\le m}(OPT_i + OPT_{i+1}) \le 2 \cdot OPT$, there exists one r with $0 \le r < k$ such that $\sum_{0\le j<\lfloor \frac{m}{k} \rfloor}(OPT_{jk+r} + OPT_{jk+r+1}) \le \frac{2}{k} \cdot OPT$. For each $0 \le j \le \lfloor \frac{m}{k} \rfloor + 1$, let G_j be the graph induced by vertices between level $(j-1)k+r$ and $jk+r+1$. In addition, we set the demands of vertices at level $(j-1)k+r$ and level $jk+r+1$ to be zero for each G_j. Clearly, the treewidth of each G_j is upper bounded by $k+1$ and the sum of the optimal cost for each G_j is no more than $(1 + \frac{4}{k}) \cdot OPT$. Take $k' = k - 1$ and we're done.

5 Concluding Remarks

We conclude with a few open problems and future research goals. First, although exact FPT algorithms are provided, the problem of approximating the optimal solution when parameterized by treewidth remains open. It would be nice to obtain faster approximation algorithms for graphs of bounded treewidth as this would provide faster approximations for planar graphs as well. Second, it would be nice to know how the problem behaves on special graph classes. As this problem has been shown to be difficult and admit a PTAS on trees when the demand can be split, approximations for other classes such as interval graphs remain unknown. Third, from the perspective of parameterization, it may be possible to find other parameters that are more closely related to the problem and obtain better results.

References

1. Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T., Niedermeier, R.: Fixed parameter algorithms for dominating set and related problems on planar graphs. Algorithmica 33(4), 461–493 (2002)
2. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial-time data reduction for dominating set. J. ACM 51(3), 363–384 (2004)

3. Baker, B.S.: Approximation algorithms for np-complete problems on planar graphs. J. ACM 41(1), 153–180 (1994)
4. Bodlaender, H.L., Koster, A.M.C.A.: Combinatorial optimization on graphs of bounded treewidth. The Computer Journal 51(3) (2008)
5. Chvátal, V.: A greedy heuristic for the set-covering problem. Mathematics of Operations Research 4(3), 233–235
6. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. J. ACM 52(6), 866–893 (2005)
7. Dom, M., Lokshtanov, D., Saurabh, S., Villanger, Y.: Capacitated domination and covering: A parameterized perspective. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 78–90. Springer, Heidelberg (2008)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
9. Feige, U.: A threshold of ln n for approximating set cover. J. ACM 45(4), 634–652 (1998)
10. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
11. Fomin, F.V., Thilikos, D.M.: Dominating sets in planar graphs: Branch-width and exponential speed-up. SIAM J. Comput. 36(2), 281–309 (2006)
12. Guo, J., Niedermeier, R.: Linear problem kernels for np-hard problems on planar graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 375–386. Springer, Heidelberg (2007)
13. Haynes, T.W., Hedetniemi, S.M., Hedetniemi, S.T., Henning, M.A.: Domination in graphs applied to electric power networks. SIAM J. Discret. Math. 15(4), 519–529 (2002)
14. Haynes, T.W., Hedetniemi, S., Slater, P.: Fundamentals of Domination in Graphs (Pure and Applied Mathematics). Marcel Dekker, New York (1998)
15. Hochbaum, D.S.: Approximation algorithms for the set covering and vertex cover problems. SIAM Journal on Computing 11(3), 555–556 (1982)
16. Hopcroft, J., Tarjan, R.: Efficient planarity testing. J. ACM 21(4), 549–568 (1974)
17. Johnson, D.S.: Approximation algorithms for combinatorial problems. In: STOC 1973: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing, pp. 38–49. ACM Press, New York (1973)
18. Kao, M.-J., Chen, H.-L.: Approximation algorithms for the capacitated domination problem (manuscript) (2010), http://arxiv.org/abs/1004.2839
19. Kao, M.-J., Liao, C.-S., Lee, D.T.: Capacitated domination problem. Algorithmica (2009)
20. Kloks, T.: Treewidth, Computations and Approximations. LNCS, vol. 842. Springer, Heidelberg (1994)
21. Liao, C.-S., Lee, D.-T.: Power domination problem in graphs. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 818–828. Springer, Heidelberg (2005)
22. Lovász, L.: On the ratio of optimal integral and fractional covers (1975)
23. Niedermeier, R.: Invitation to Fixed Parameter Algorithms. Oxford University Press, Oxford (2006)
24. Roberts, F.S.: Graph Theory and Its Applications to Problems of Society (1978)
25. Wan, P.-J., Alzoubi, K.M., Frieder, O.: A simple heuristic for minimum connected dominating set in graphs. International Journal of Foundations of Computer Science 14(2), 323–333 (2003)

A Polynomial Time Approximation Scheme for Embedding Hypergraph in a Weighted Cycle

Chaoxia Yang and Guojun Li*

School of Mathematics, Shandong University
Jinan 250100, China
Department of Biochemistry and Molecular Biology
University of Georgia, GA 30602, USA
{chaoxia,guojun}@csbl.bmb.uga.edu

Abstract. The problem of Minimum Congestion Hypergraph Embedding in a Weighted Cycle (MCHEWC) is to embed the hyperedges of a hypergraph as paths in a weighted cycle such that the maximum congestion, i.e. the maximum product of the weight of an edge and the number of times that the edge is passed by the embedding, is minimized. It is known that the problem, the same as the unweighted case, is NP-hard. The aim of this paper is to present a polynomial time approximation scheme (PTAS) for the problem.

Keywords: hypergraph embedding, weighted cycle, minimum congestion, NP-hard, polynomial time approximation scheme.

1 Introduction

The problem of Minimum-Congestion Hypergraph Embedding in a Cycle (denoted by MCHEC) was proposed by Ganley and Cohoon [3]. In that problem, we wish to embed the m-hyperedges of an n-vertex hypergraph as paths in an n-vertex cycle, such that the maximum number of paths that pass through an edge in the cycle is minimized. The problem of Minimum-Congestion Hypergraph Embedding in a Weighted Cycle (MCHEWC) is a weighted version of the previous problem of which each cycle is considered as weighted and hence different. The congestion of an edges now is defined as the number of paths that pass through it times its weight. The aim is also to find an embedding such that the maximum congestion is minimized. The MCHEWC problem is a challenging problem with many applications in electronic design automation, computer networks, parallel computing and computer communication.

Several studies related to the MCHEWC problem have been performed. The optimal solution for the problem of Minimum Congestion Graph Embedding in

* To this author the correspondence should be addressed. This work is supported by National Science Foundation of China under Grant No.fun 60673059, 10631070 and 60873207, and also by endowed fund from Shandong Province of China.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 197–209, 2010.

a Cycle (MCGEC) is solved in polynomial time by an algorithm developed by Frank et al. [1,2]. Ganley and Cohoon [3] showed that the MCHEC problem is NP-hard in general, but solvable in polynomial time when the congestion is at most k, where k is a fixed number. In that case, a solution can be computed in $O((nm)^{k+1})$ time. Other approximation algorithms have been proposed subsequently. Gu and Wang [4] presented an algorithm for solving the MCHEC problem with an approximation ratio 1.8 which was further reduced to 1.5 by Lee and Ho [6]. Very recently, Li et al. [7] presented a polynomial time approximation scheme for solving the MCHEC problem. Lee and Ho [5] showed that the problem of Weighted Hypergraph Embedding in a Cycle (WHEC) and the problem of Weighted Graph Embedding in a Cycle (WGEC) are both NP-hard and proposed a 2-approximation algorithm using the idea of removing the longest adjacent paths. In this paper, we present for the first time a polynomial time approximation scheme to the MCHEWC problem.

2 Preliminaries

In this section we introduce some notations which will be used in this paper. A weighted cycle C of n nodes is an undirected graph $G = (V, E_G)$ with node set $V = \{i | 1 \leq i \leq n\}$ and the weighted edge set $E_G = \{e_i | 1 \leq i \leq n\}$, where e_i represents the edge of weight w_i connecting the nodes i and $i + 1$ for $i = 1, 2, ..., n$. Throughout this paper, we assume that all nodes are uniquely labelled using nonnegative integers after performing modulo n operation. Without loss of generality, we assume that the numbers on the nodes are ordered in the clockwise direction. A hypergraph $H = (V, E_H)$ is defined over the same node set $V = \{i | 1 \leq i \leq n\}$ with the set $E_H = \{h_1, h_2, ..., h_m\}$ of the hyperedges, where each hyperedge h_j is a subset of V with two or more nodes.

For each $j (1 \leq j \leq m)$, we define a *connecting path* (or c-path) P_j for hyperedge h_j, where P_j is a minimal path in the weighted cycle C containing all nodes in h_j. Therefore, there are exactly $|h_j|$ possible c-paths for each hyperedge h_j. Choosing one c-path for each hyperedge of H, we have an embedding of the hypergraph H in a weighted cycle that is a set of c-paths in C. Given an embedding of a hypergraph, the congestion of each edge of C is the product of its weight and the number of c-paths that contain the edge. For a given hypergraph and a weighted cycle on the same node set, the MCHEWC problem is to find an embedding of the hypergraph such that the maximum congestion in the weighted cycle is minimized.

Formally, we introduce the following notation. For each $j (1 \leq j \leq m)$, let hyperedge $h_j = \{i_1^j, i_2^j, ..., i_{|h_j|}^j\}$ be such that nodes, $i_1^j, i_2^j, ...,$ and $i_{|h_j|}^j$, are ordered in the clockwise order along the cycle C. Then h_j partitions the cycle C into $|h_j|$ segments: $E_l^j, l = 1, 2, ..., |h_j|$, where $E_l^j = \{e_i | i_l^j \leq i \leq i_{l+1}^j - 1\}$. Note that the arithmetic operations involving the subscripts of the indices are performed by modulo $|h_j|$.

A c-path is called an E_l^j-*embedding* for hyperedge h_j if the c-path consists of edges of $E_G \setminus E_l^j$. An embedding of the hypergraph is a set of m c-paths, one c-path for each hyperedge. There are $|h_j|$ different embeddings for each hyperedge h_j and the total number of feasible solutions to the MCHEWC problem is therefore $|h_1||h_2|...|h_m|$.

We use a vector $x = (x_1, x_2, ..., x_m)$ to denote a solution, where x_j represents an $E_{l_j}^j$-*embedding* of the hyperedge h_j for some $l_j (1 \leq l_j \leq |h_j|)$, and call it an x-*embedding* of the hypergraph. For an edge e_i in the cycle C, we use $e_i(x)$ to denote the number of c-paths passing through the edge e_i regarding the x-*embedding*. Then the MCHEWC problem can be modeled as the following optimization problem.

$$\begin{cases} \min \ \ z; \\ w_i e_i(x) \leq z, \ \ i = 1, 2, \ldots, n. \end{cases} \tag{1}$$

Since the problem is known to be NP-complete, our interest is in the algorithms which return solutions with guaranteed near-optimum values. By $\mathrm{OPT}(I)$ we denote the optimum objective value on instance I of the MCHEWC problem and by $\mathrm{A}(I)$ the cost of the solution given by an algorithm A. We want to find a polynomial time approximation scheme (PTAS) to solve the MCHEWC problem. That is, we want to find an algorithm which has the following performance ratio

$$R_A(I, \epsilon) = \frac{A(I)}{OPT(I)} \leq 1 + \epsilon$$

for any given constant $\epsilon > 0$.

Given a weighted cycle C, we take $w_i := w_i / w_{max} (1 \leq i \leq n)$, where $w_{max} = \max\{w_i | 1 \leq i \leq n\}$. Then the new embedding problem is equivalent to the original embedding problem. For the sake of convenience, from now on, we may assume that each edge in the cycle C has a weight belonging to $(0, 1]$.

3 Development of PTAS for Special Cases

In this section, we develop PTAS for some special cases. In the first place, we consider the special case that the number of hyperedges is bounded from above by $c \log(n)$ with c a positive constant, we have the following lemma.

Lemma 1. *Let $r (1 \leq r \leq n)$ be an integer and $x=(x_1, x_2, ..., x_m)$ an embedding of H where x_j is an $E_{l_j}^j$-embedding of h_j. Define*

$$\Omega_{i_1, i_2, ..., i_r}(x) = \{j \mid E_{l_j}^j \cap \{e_{i_1}, e_{i_2}, ..., e_{i_r}\} \neq \emptyset\}$$

for any indices $1 \leq i_1, i_2, ..., i_r \leq n$. Then there exist r indices $1 \leq m_1, m_2, ..., m_r \leq n$ such that for any embedding $x' = (x_1', x_2', ..., x_m')$ of H which satisfies $x_j' = x_j$ for all $j \in \Omega_{m_1, m_2, ..., m_r}(x)$, the following inequality

$$w_i e_i(x') - w_i e_i(x) \leq \frac{1}{r} w_i e_i(x)$$

holds for all i.

Proof. Let $1 \leq i_1, i_2, ..., i_r \leq n$ be r distinct indices of edges on C and $x = (x_1, x_2, ..., x_m)$ an embedding of H. We use these edges $e_{i_1}, e_{i_2}, ..., e_{i_r}$ and the given embedding x to create a set Ω as follows.

$$\Omega_{i_1,i_2,...,i_r}(x) = \{j|\ E^j_{l_j} \cap \{e_{i_1}, e_{i_2}, ..., e_{i_r}\} \neq \emptyset\}.$$

Notice that for the fixed x and r, $\Omega_{i_1,i_2,...,i_r}(x)$ depends on the choice of indices, $i_1, i_2, ..., i_r$ of edges on C. By $\Omega(x, r)$ we denote the one of maximum size which is reached at $m_1, m_2, ..., m_r$, i.e. $\Omega(x, r) = \Omega_{m_1,m_2,...,m_r}(x)$. We will show that $\Omega(x, r)$ is the one required to make the lemma true. To this end, we define

$$\Omega(i) = \{j \mid e_i \in E^j_{l_j} \text{ and } e_{m_h} \notin E^j_{l_j} \text{ for } h = 1, 2, \ldots, r\}.$$

It follows from the definition that

$$\Omega(i) = \Omega_{m_1,m_2,...,m_r,i}(x) - \Omega_{m_1,m_2,...,m_r}(x) \tag{2}$$

For $1 \leq t \leq r$, we define

$$\Omega(m_t, i) = \{j \mid e_{m_t} \in E^j_{l_j} \text{ and } e_i, e_{m_h} \notin E^j_{l_j} \text{ for } h \in \{1, 2, \ldots, r\} - \{t\}\}.$$

From the definition again, we get

$$\Omega(m_t, i) = \Omega_{m_1,m_2,...,m_r,i}(x) - \Omega_{m_1,...,m_{t-1},m_{t+1},...,m_r,i}(x) \tag{3}$$

The maximality of $\Omega_{m_1,m_2,...,m_r}(x)$ together with (2) and (3) make sure that

$$|\Omega(m_t, i)| \geq |\Omega(i)|.$$

We now further claim that $\Omega(m_t, i), t = 1, 2, ..., r$ does not overlap to each other. For, suppose on the contrary, there are two integers $p, q (1 \leq p, q \leq r)$ and $j \in \Omega(m_p, i) \cap \Omega(m_q, i)$. It follows from the definition of $\Omega(m_t, i)$ that

$$e_{m_p} \in E^j_{l_j} \text{ and } e_i, e_{m_h} \notin E^j_{l_j}, h \in \{1, 2, \ldots, r\} - \{p\},$$

and simultaneously that

$$e_{m_q} \in E^j_{l_j} \text{ and } e_i, e_{m_h} \notin E^j_{l_j}, h \in \{1, 2, \ldots, r\} - \{q\},$$

which obviously contradict each other.

Keeping in mind that the embedding x_j of hyperedge h_j contains e_i if and only if $e_i \notin E^j_{l_j}$, we see that the embedding of each hyperedge h_j with $j \in \Omega(m_t, i)$ contains the edge e_i. This observation together with the above claim ensure that

$$e_i(x) \geq \sum_{t=1}^{r} |\Omega(m_t, i)| \geq r|\Omega(i)| \tag{4}$$

Now we are ready to prove the lemma. Let x' be an embedding of H such that $x'_j = x_j$ for all $j \in \Omega(x,r)$, and l'_j be the index such that x'_j is a $E^j_{l'_j}$−embedding of h_j. Then we have

$$\begin{aligned} e_i(x') - e_i(x) &\le |\{j \mid e_i \notin E^j_{l'_j} \text{ and } e_i \in E^j_{l_j}\}| \\ &= |\{j \notin \Omega(x,r) \mid e_i \notin E^j_{l'_j} \text{ and } e_i \in E^j_{l_j}\}| \\ &\le |\{j \notin \Omega(x,r) \mid e_i \in E^j_{l_j}\}| \\ &= |\{j \mid e_i \in E^j_{l_j} \text{ and } e_{m_h} \notin E^j_{l_j}, h = 1,2,\ldots,r\}| \\ &= |\Omega(i)| \le \frac{1}{r} e_i(x) \quad \text{(from (4))}. \end{aligned}$$

Since $w_i > 0$ for all $i = 1, 2, ..., n$, we obtain $w_i e_i(x') - w_i e_i(x) \le \frac{1}{r} w_i e_i(x)$.

Similar to [7], Lemma 1 leads to a polynomial time approximation scheme to the MCHEWC problem under the condition when $m \le c\log(n)$. For any prespecified real number $\varepsilon > 0$, we take an integer r such that $1/r < \varepsilon$. Notice that if we replace x by an optimal solution x^* in Lemma 1 then x' described there is an approximation within a factor of $1 + \varepsilon$ of the optimum. Now we demonstrate that such an approximation scheme x' can be accomplished in a polynomial time.

Note that for fixed x and r, $\Omega_{i_1,i_2,\ldots,i_r}(x)$ depends on the choice of the indices, $i_1, i_2, ..., i_r$, of edges on C. By $\Omega(x,r)$ we denote the one of maximum size which is reached at $m_1, m_2, ..., m_r$, i.e. $\Omega(x,r) = \Omega_{m_1,m_2,\ldots,m_r}(x)$. Suppose we are lucky that $m_1, m_2, ..., m_r$ appear to us such that $\Omega(x^*,r) = \Omega_{m_1,m_2,\ldots,m_r}(x^*)$ because it is allowed to enumerate all $\binom{n}{r}$ possibilities. For each $j (1 \le j \le m)$, let

$$E^j = \{E^j_l | E^j_l \cap \{e_{m_1}, e_{m_2}, ..., e_{m_r}\} \ne \emptyset, 1 \le l \le |h_j|\},$$

then $|E^j| \le \min\{r, |h_j|\}$ for any j. For each j, we enumerate all $|E^j|$ possible embeddings to form a set of solutions which is denoted by X, then we have $|X| \le \Pi_j |E^j| \le r^m \le n^{c \ln r}$. Lemma 1 makes sure that X contains an embedding x' such that $w_i e_i(x') \le (1+\epsilon) w_i e_i(x)$, so the approximation scheme x' can be accomplished within the time up bounded by $\binom{n}{r} n^{c \ln r}$. The algorithm in this case is described schematically as follows.

Algorithm. *specialEmbedding:*
Input: $G = (V, E_G)$ and $H = (V, E_H)$.
Output: an x-embedding of H.
1. **for** each r-element subset $\{e_{i_1}, e_{i_2}, ..., e_{i_r}\}$ of the n input edges in E_G **do**
   (a) $E^j = \{E^j_l | E^j_l \cap \{e_{i_1}, e_{i_2}, ..., e_{i_r}\} \ne \emptyset, 1 \le l \le |h_j|\}$
   (b) Create a set of solutions by enumerating all $|E^j|$ possible embeddings of h_j
2. **Output** the best solution obtained in Step 1.

Theorem 1. *The MCHEWC problem can be solved with a PTAS when* $m \leq c \log n$*, where* c *is a constant independent of* m *and* n*.*

Now we restrict the problem to the case when $m > c \log n$. We define a variable $x_{j,l}$ to be one if x_j is the E_l^j-embedding and zero otherwise for $1 \leq j \leq m$ and $1 \leq l \leq |h_j|$; and further an index function $\chi_j(e_i, l)$ to be zero if $e_i \in E_l^j$ and one if $e_i \notin E_l^j$. Then (1) is equivalent to following 0-1 optimization problem.

$$\begin{cases} \min \quad z; \\ \sum_{l=1}^{|h_j|} x_{j,l} = 1, \quad j = 1, 2, \ldots, m, \\ w_i \sum_{j=1}^{m} \sum_{l=1}^{|h_j|} \chi_j(e_i, l) x_{j,l} \leq z, \quad i = 1, 2, \ldots, n. \end{cases} \tag{5}$$

Let $\bar{x}_{j,l}, j = 1, 2, ..., m; l = 1, 2, ..., |h_j|$ be its fractional optimal solution, and

$$\tau_i = w_i \sum_{j=1}^{m} \sum_{l=1}^{|h_j|} \chi_j(e_i, l) \bar{x}_{j,l}.$$

Hereafter, by c_{opt} we denote the optimum value of (5), then $\tau_i \leq c_{opt}$ for all i. In this section, when $m > c \log n$, we restrict ourselves to the case $c_{opt} \geq dm$ for some constant $d (0 < d \leq 1)$. We apply a standard randomized rounding strategy to the fractional optimal solution $\bar{x}_{j,l}, j = 1, 2, ..., m; l = 1, 2, ..., |h_j|$. For each $j = 1, 2, ..., m$, independently, with probability $\bar{x}_{j,l}$, set $x'_{j,l} = 1$ and $x'_{j,h} = 0$ for any $h \in \{1, 2, ..., |h_j|\} - \{l\}$. Then we get a 0-1 solution $x'_{j,l}, j = 1, 2, ..., m; l = 1, 2, ..., |h_j|$ to (5), hence a solution to (1).

Lemma 2. *Let* $X_1, X_2, ..., X_n$ *be* n *independent random* 0-1 *variables, where* X_i *takes* 1 *with probability* $p_i, 0 < p_i < 1$*. Let* $X = \sum_{i=1}^{n} a_i X_i, 0 < a_i \leq 1$*, and* $\mu = E[X]$*. Then for any* $0 < \delta \leq 1/2$, $\boldsymbol{Pr}(X > \mu + \delta n) < exp\{-\frac{1}{3} n \delta^2\}$.

Proof. Let $X' = \sum_{i=1}^{n} X_i$, and $\mu' = E[X']$. Then $\mu' = \sum_{i=1}^{n} p_i \leq n$, $\mu = \sum_{i=1}^{n} a_i p_i \leq n$, and $\mu' \geq \mu$. To prove this lemma, we need to resort to the following Markovian inequality:

$$\mathbf{Pr}(X > \mu + \delta n) = \mathbf{Pr}(e^{tX} > e^{t(\mu + \delta n)}) \leq \frac{E[e^{tX}]}{e^{t(\mu + \delta n)}}$$

for $\forall t > 0$. Note that

$$E[e^{t a_i X_i}] = p_i e^{t a_i} + (1 - p_i) = 1 + p_i (e^{t a_i} - 1) \leq e^{p_i (e^{t a_i} - 1)}.$$

Therefore,

$$\begin{aligned} E[e^{tX}] = E[e^{\sum_{i=1}^{n} t a_i X_i}] &= \prod_{i=1}^{n} E[e^{t a_i X_i}] \leq \prod_{i=1}^{n} e^{p_i (e^{t a_i} - 1)} \\ &= exp\{\sum_{i=1}^{n} p_i (e^{t a_i} - 1)\} = exp\{\sum_{i=1}^{n} p_i e^{t a_i} - \sum_{i=1}^{n} p_i\} \\ &= exp\{\sum_{i=1}^{n} p_i e^{t a_i} - \mu'\}. \end{aligned}$$

Let $t = \ln(1+\delta) > 0 \quad (\forall \delta > 0)$, we then get

$$\mathbf{Pr}(X > \mu + \delta n) \leq \frac{e^{\sum_{i=1}^{n} p_i e^{ta_i} - \mu'}}{e^{t(\mu+\delta n)}} = \frac{e^{\sum_{i=1}^{n} p_i (1+\delta)^{a_i} - \mu'}}{(1+\delta)^{(\mu+\delta n)}} = (\frac{e^{\frac{\sum_{i=1}^{n} p_i (1+\delta)^{a_i} - \mu'}{n}}}{(1+\delta)^{(\frac{\mu}{n}+\delta)}})^n.$$

Now we only need to prove that for $0 < \delta \leq \frac{1}{2}$,

$$\frac{e^{\frac{\sum_{i=1}^{n} p_i (1+\delta)^{a_i} - \mu'}{n}}}{(1+\delta)^{(\frac{\mu}{n}+\delta)}} \leq e^{-\delta^2/3}.$$

After taking logarithm on the both sides of above inequality, we get that

$$\frac{\sum_{i=1}^{n} p_i (1+\delta)^{a_i} - \mu'}{n} - (\frac{\mu}{n} + \delta)\ln(1+\delta) + \frac{\delta^2}{3} \leq 0.$$

Let $f(\delta) = \frac{\sum_{i=1}^{n} p_i (1+\delta)^{a_i} - \mu'}{n} - (\frac{\mu}{n} + \delta)\ln(1+\delta) + \frac{\delta^2}{3}$, we only need to prove $f(\delta) \leq 0$ for $0 < \delta \leq \frac{1}{2}$.

By simple calculation of first and second derivation of $f(\delta)$, we get

$$f'(\delta) = \frac{\sum_{i=1}^{n} p_i a_i (1+\delta)^{a_i - 1}}{n} - \frac{\frac{\mu}{n} + \delta}{1+\delta} - \ln(1+\delta) + \frac{2}{3}\delta,$$

$$f''(\delta) = \frac{\sum_{i=1}^{n} p_i a_i (a_i - 1)(1+\delta)^{a_i - 2}}{n} + \frac{\frac{\mu}{n} - 1}{(1+\delta)^2} - \frac{1}{1+\delta} + \frac{2}{3}.$$

Since $f''(0) \leq 0$, $f'(\delta)$ must be nonincreasing in $(0, \frac{1}{2}]$, and thus $f'(\delta) \leq 0$ due to $f'(0) = 0$. Therefore $f(\delta)$ is nonincreasing, along with $f(0) = 0$, it is finally ensured that $f(\delta) \leq 0$ in $(0, \frac{1}{2}]$.

When $m \geq c \log n$ and $c_{opt} \geq dm$, we have

Lemma 3. *Let $x_{j,\,l}$ be a 0-1 solution to (5) after randomized rounding procedure. Then for any fixed small number $\epsilon > 0$, with probability at least $1 - n^{1-\frac{1}{3}\epsilon^2 d^2 c}$,*

$$w_i e_i(x) \leq (1+\epsilon) c_{opt},$$

for each $e_i \in E_G$.

Proof. Recalling that for each j, $x_{j,l}$ is rounded to 1 only for one index l ($1 \leq l \leq |h_j|$), we see that the variable $\sum_{l=1}^{|h_j|} \chi_j(e_i, l) x_{j,l}$ rounds to either 1 or 0, and is independent to each other for different j's. Let

$$w_i e_i(x) = w_i \sum_{j=1}^{m} \sum_{l=1}^{|h_j|} \chi_j(e_i, l) x_{j,l}, 0 < w_i \leq 1.$$

The expectation of the sum of variables is

$$E[w_i e_i(x)] = w_i \sum_{j=1}^{m} \sum_{l=1}^{|h_j|} \chi_j(e_i, l)\overline{x}_{j,l} = \tau_i \leq c_{opt}.$$

For any fixed $\delta > 0$, applying Lemma 2 to $w_i e_i(x)$, we get

$$\mathbf{Pr}(w_i e_i(x) > \tau_i + \delta m) < e^{-\delta^2 m/3},$$

and hence

$$\mathbf{Pr}(w_i e_i(x) > \tau_i + \delta m \ \text{ for at least one } i) < n \times e^{-\delta^2 m/3}.$$

By utilizing the assumption $m \geq c \log n$, we obtain

$$\mathbf{Pr}(w_i e_i(x) \leq \tau_i + \delta m \ \text{ for all } i) \geq 1 - n^{1-\delta^2 c/3}.$$

If we take $\delta = d\epsilon$, we get $\tau_i + \delta m \leq (1+\epsilon)c_{opt}$. The lemma is proved.

Corollary 1. *For $\delta = d\epsilon$ and $c > 3/\delta^2$ be a constant number. If $m > c \log n$ then there exists an embedding x such that $w_i e_i(x) \leq (1+\epsilon)c_{opt}$ for all i.*

Proof. The existence of such an embedding obtained from the proof of Lemma 3 because $c > 3/\delta^2$, making sure that

$$\mathbf{Pr}(w_i e_i(x) \leq (1+\epsilon)c_{opt}) \geq \mathbf{Pr}(w_i e_i(x) \leq \tau_i + \delta m \ \text{ for all } i) > 0.$$

Recall that $\bar{x}_{j,l}$ is a fractional solution to (5). We partition $\{1, 2, ..., m\}$ into $L_1, L_2, ..., L_k$ such that $c \log n \leq |L_h| < 2c \log n$ for $h = 1, 2, ..., k$. Let $\tau_{i,h} = w_i \sum_{j \in L_h} \sum_{l=1}^{|h_j|} \chi(e_i, l)\bar{x}_{j,\,l}$. For each h, Corollary 1 guarantees that there exists an embedding x_h for the hyperedge h_j with $j \in L_h$ such that

$$w_i e_i(x_h) \leq \tau_{i,h} + \delta |L_h|$$

holds for all i.

Applying *specialEmbedding* to $L_h, h = 1, 2, ..., k$, we can find an approximation x'_h in polynomial time such that $w_i e_i(x'_h) - w_i e_i(x_h) \leq \delta w_i e_i(x_h)$ for all i. Therefore,

$$w_i e_i(x'_h) \leq \tau_{i,h} + \delta |L_h| + \delta w_i e_i(x_h)$$

holds for all i.

Let x' be a concatenation of x'_hs. It follows that

$$\begin{aligned} w_i e_i(x') &\leq \sum_{h=1}^{k} (\tau_{i,h} + \delta |L_h| + \delta w_i e_i(x_h)) = \tau_i + \delta m + \delta w_i e_i(x) \\ &\leq \tau_i + \delta m + \delta \tau_i + \delta^2 m \leq [1 + (1 + \frac{2}{d})\delta] c_{opt} \leq (1+\epsilon) c_{opt}, \end{aligned}$$

where $\epsilon = (1 + 2/d)\delta$.

Theorem 2. *The MCHEWC problem can be solved with a PTAS when* $c_{opt} \geq dm(0 < d \leq 1)$ *and* $m > c \log n$*, where* c *is a constant number.*

4 An Ultimate PTAS

When the optimal congestion c_{opt} is small relative to the number of hyperedges, randomized rounding and derandomization scheme does not necessarily work to the MCHEWC problem. To overcome this obstacle, for any given $\epsilon > 0$ we collect all the edges from E_G of weight no less than ϵ getting a set of larger edges denoted by K, i.e. $K = \{e_i | w_i \geq \epsilon\}$. For an integer $r \leq |K|$, and edges $e_{i_1}, e_{i_2}, ..., e_{i_r}$ in K, we define

$$R_{i_1,i_2,...,i_r} = \{j \,|\, e_{i_1}, e_{i_2}, ..., e_{i_r} \in E_l^j \text{ for some } l(1 \leq l \leq |h_j|)\}$$

$$U_{i_1,i_2,...,i_r} = \{1, 2, ..., m\} - R_{i_1,i_2,...,i_r}.$$

The following lemma makes the techniques developed in the last section be applicable to $U_{i_1,i_2,...,i_r}$.

Lemma 4. $c_{opt} \geq \frac{\epsilon}{r}|U_{i_1,i_2,...,i_r}|$.

Proof. For each $j \in U_{i_1,i_2,...,i_r}$, it follows from the definition that these r edges, $e_{i_1}, e_{i_2}, ..., e_{i_r}$ must belong to different segments $E_l^j, l = 1, 2, ..., |h_j|$, of jth hyperedge. Let $x = \{x_1, x_2, ..., x_m\}$ be an optimum solution where x_j represents an $E_{l_j}^j$-embedding. Since $\{e_{i_1}, e_{i_2}, ..., e_{i_r}\} \not\subset E_{l_j}^j$ there exists $h(1 \leq h \leq r)$ such that $e_{i_h} \in x_j$, implying that the embedding x_j for $j \in U_{i_1,i_2,...,i_r}$ must contribute one to $e_{i_h}(x)$. However the optimality of x implies that $\epsilon e_{i_h}(x) \leq w_{i_h} e_{i_h}(x) \leq c_{opt}$, which further guarantees that $|U_{i_1,i_2,...,i_r}|\epsilon \leq r c_{opt}$. This proves the lemma.

Lemma 4 guarantees that it can be approximated on $U_{i_1,i_2,...,i_r}$ for $e_{i_1}, e_{i_2}, ..., e_{i_r}$ in K by using the the techniques developed in last section. Now we are going to show that it is able to be approximated either on $R_{i_1,i_2,...,i_r}$ with special selection of $e_{i_1}, e_{i_2}, ..., e_{i_r}$. To this end, we use x to denote a putative optimum solution of objective value c_{opt}. Let y be an embedding of H and P a subset of indices of hyperedges of H. We use $y|_P$ to denote a partial embedding of y whose component indices are restricted in P. The definition of $R_{i_1,i_2,...,i_r}$ implies that for each $j \in R_{i_1,i_2,...,i_r}$, there is an integer l_j such that $e_{i_1}, e_{i_2}, ..., e_{i_r} \in E_{l_j}^j$. Let x_j^* be the $E_{l_j}^j$-embedding for all $j \in R_{i_1,i_2,...,i_r}$, and $x^*|_{R_{i_1,i_2,...,i_r}}$ be a partial embedding vector having component x_j^* for all $j \in R_{i_1,i_2,...,i_r}$. We will show that there exist indices $i_1, i_2, ..., i_r$ such that the partial embedding vector $x^*|_{R_{i_1,i_2,...,i_r}}$ does form a good approximation to the optimum embedding x on $R_{i_1,i_2,...,i_r}$.

Let $x = (x_1, x_2, \ldots, x_m)$ be an optimal embedding such that $\max_{1 \leq i \leq n} w_i e_i(x)$ is minimized, and l the index such that $e_l(x) = \min\{e_i(x) | \, e_i(x) \neq 0, 1 \leq i \leq n\}$. Define

$$p_{i_1,i_2,...,i_k} = |\{j \in R_{i_1,i_2,...,i_k} | x_j^* \neq x_j\}|$$

and

$$\rho_k = \min_{1 \le i_1, i_2, \ldots, i_k \le n} \frac{p_{l,i_1,i_2,\ldots,i_k}}{e_l(x)}.$$

Lemma 5. *There are edges $e_{i_1}, e_{i_2}, ..., e_{i_r}$ in K, such that for any $e_i \in E_G$,*

$$w_i e_i(x^*|_{R_{l,i_1,i_2,\ldots,i_r}}) - w_i e_i(x|_{R_{l,i_1,i_2,\ldots,i_r}}) \le \epsilon c_{opt}.$$

Proof. The proof is divided into two cases: 1) $|K| \ge \frac{1}{\epsilon}$; 2) $|K| < \frac{1}{\epsilon}$.
1) In this case the integer r can be set to be greater than or equal to $\frac{1}{\epsilon}$ since $|K| \ge \frac{1}{\epsilon}$. Note that $p_{l,i_1} = |\{j \in R_{l,i_1} | x_j^* \ne x_j\}|$ is the congestion of e_l caused by x_j for $j \in R_{l,i_1}$. Therefore, $p_{l,i_1} \le e_l(x)$. It follows from the definition that $\rho_1 \le 1$. In addition, ρ_k is non-increasing as k increases. Since the sum of r terms $(\rho_1 - \rho_2) + (\rho_2 - \rho_3) + ... + (\rho_r - \rho_{r+1}) = \rho_1 - \rho_{r+1} \le \rho_1 \le 1$ there is $k (1 \le k \le r)$ such that $\rho_k - \rho_{k+1} \le \frac{1}{r}$. For all $e_i \in E_G$, we define

$$R(i) = \{j \in R_{l,i_1,i_2,\ldots,i_r} | x_j^* \ne x_j \text{ and } x_j^* \ne x_j^i\}$$

where x_j^i represents the E_l^j-embedding such that $e_i \in E_l^j$. Then it follows

$$e_i(x^*|_{R_{l,i_1,i_2,\ldots,i_r}}) - e_i(x|_{R_{l,i_1,i_2,\ldots,i_r}}) \le |R(i)|.$$

To prove the lemma, we only need to examine that $w_i|R(i)| \le \frac{1}{r} c_{opt}$ since $\frac{1}{r} \le \epsilon$ in this case. To do so, we fix the indices $1 \le i_1, i_2, ..., i_k \le n$ such that $p_{l,i_1,i_2,\ldots,i_k} = \rho_k e_l(x)$. Then for any $r (k < r < n)$ and $i (1 \le i \le n)$, we have

$$\begin{aligned}
|R(i)| &= |\{j \in R_{l,i_1,i_2,\ldots,i_r} | x_j^* \ne x_j \text{ and } x_j^* \ne x_j^i\}| \\
&\le |\{j \in R_{l,i_1,i_2,\ldots,i_k} | x_j^* \ne x_j \text{ and } x_j^* \ne x_j^i\}| \\
&= |\{j \in R_{l,i_1,i_2,\ldots,i_k} | x_j^* \ne x_j\}| - |\{j \in R_{l,i_1,i_2,\ldots,i_k} | x_j^* = x_j^i \text{ and } x_j^* \ne x_j\}| \\
&= |\{j \in R_{l,i_1,i_2,\ldots,i_k} | x_j^* \ne x_j\}| - |\{j \in R_{l,i_1,i_2,\ldots,i_k,i} | x_j^* \ne x_j\}| \\
&= p_{l,i_1,i_2,\ldots,i_k} - p_{l,i_1,i_2,\ldots,i_k,i} \le \rho_k e_l(x) - \rho_{k+1} e_l(x) \le \frac{e_l(x)}{r}.
\end{aligned}$$

It turns out that

$$w_i|R(i)| \le \frac{w_i e_l(x)}{r} \le \frac{w_i e_i(x)}{r} \le \frac{1}{r} c_{opt}.$$

2) Let $K = \{e_{i_1}, e_{i_2}, ..., e_{i_r}\}$ and $r < \frac{1}{\epsilon}$. By T we denote the subset of $R_{i_1,\ldots,i_r}$ such that $x_j \ne x_j^*$ for each $j \in T$, and thus the hyperedge h_j must be embedded through the edge $e_k \in K$ with $w_k = 1$ in the partial optimum embedding $x|_T$. It is obvious that $|T| \le c_{opt}$, and that

$$w_i e_i(x^*|_{R_{i_1,i_2,\ldots,i_r}}) - w_i e_i(x|_{R_{i_1,i_2,\ldots,i_r}}) = w_i e_i(x^*|_T) - w_i e_i(x|_T).$$

Notice that

$$w_i e_i(x^*|_T) - w_i e_i(x|_T) = -w_i|T|$$

when $e_i \in \{e_{i_1}, e_{i_2}, ..., e_{i_r}\}$, and

$$w_i e_i(x^*|_T) - w_i e_i(x|_T) \leq w_i|T|$$

when $e_i \notin \{e_{i_1}, e_{i_2}, ..., e_{i_r}\}$. Recalling that $w_i < \epsilon$ when $e_i \notin \{e_{i_1}, e_{i_2}, ..., e_{i_r}\}$ and $|T| \leq c_{opt}$, we get

$$w_i e_i(x^*|_{R_{i_1,i_2,...,i_r}}) - w_i e_i(x|_{R_{i_1,i_2,...,i_r}}) \leq \epsilon c_{opt}.$$

The proof for the lemma is complete.

The above lemma implies that $x^*|_{R_{i_1,...,i_r}}$ does approximate the optimal embedding x as good as we want. For simplicity, we write $U = U_{i_1,...,i_r}$ and $R = \{1, 2, ..., m\} - U$. Now we focus on the hyperedges with indices in U. A combination of the ideas developed in the last section applies to this situation.

Case 1. $|U| \leq c \log n$. Using the techniques developed in section 3 , we find a partial embedding $x''|_U$ on U in polynomial time such that

$$w_i e_i(x''|_U) - w_i e_i(x|_U) \leq \epsilon c_{opt}$$

hold for any $e_i \in E_G$. By Lemma 5, the whole solution $x = (x'_1, x'_2, ..., x'_m)$ defined by

$$x'_j = \begin{cases} x^*_j, & \text{if } j \in R, \\ x''_j, & \text{if } j \in U. \end{cases}$$

satisfies

$$w_i e_i(x') \leq (1 + 2\epsilon) c_{opt}$$

for any $e_i \in E_G$.

Case 2. $|U| > c \log n$. Lemma 4 implies that the techniques developed in section 3 are applicable to U. Lemma 5 guarantees that the following optimization problem

$$\begin{cases} \min \quad z; \\ \sum_{l=1}^{|h_j|} x_{j,l} = 1, \quad j = 1, 2, \ldots, |U|, \\ w_i \sum_{j=1}^{|U|} \sum_{l=1}^{|h_j|} \chi_j(e_i, l) x_{j,l} \leq z - w_i e_i(x^*|_R), \quad i = 1, 2, \ldots, n. \end{cases} \tag{6}$$

has a fractional solution $\bar{x}_{j,l} (1 \leq j \leq |U|, 1 \leq l \leq |h_j|)$ with cost $\bar{d} \leq (1+2\epsilon) c_{opt}$. From the proof of Lemma 3, we have

Lemma 6. *Let $x'|_U$ be a 0-1 solution of (6) after randomized rounding. Then for any $\delta > 0$, with certain probability> 0,*

$$w_i e_i(x'|_U) + w_i e_i(x^*|_R) \leq \tau_i + w_i e_i(x^*|_R) + \delta|U|,$$

hold for all $e_i \in E_G$, where $\tau_i = w_i \sum_{j=1}^{|U|} \sum_{l=1}^{|h_j|} \chi_j(e_i, l) \bar{x}_{j,l}$.

Algorithm general Embedding
Input: $G = (V, E_G)$ and $H = (V, E_H)$.
Output: an x-embedding of H.
1. Calculate $K = \{e_i | w_i \geq \epsilon\}$.
If $|K| \geq \frac{1}{\epsilon}$, **then**
for each r-element subset $\{e_{i_1}, e_{i_2}, ..., e_{i_r}\} \subseteq K$ **do**
(a) $R = \{j \mid e_{i_1}, e_{i_2}, ..., e_{i_r} \in E_l^j\}$, and $U = \{1, 2, ..., m\} - R$.
(b) Let $x^*|_R$ be a partial approximation on R with x_j^* being E_l^j-embedding for $j \in R$.
(c) find a partial embedding $x'|_U$ using techniques developed in §4.
(d) concatenate $x^*|_R$ and $x'|_U$ to get an approximation x'.
else $K = \{e_{i_1}, ..., e_{i_k}\}$, $R = \{j | e_{i_1}, e_{i_2}, ..., e_{i_k} \in E_l^j\}$, $U = \{1, 2, ..., m\} - R$,
do $(b), (c)$ and (d).
2. **Output** the best solution obtained in Step 1.

Fig. 1. Outline for the ultimate PTAS

By using standard derandomization procedure used in section 3, we can find an approximation x'' on U in polynomial time such that

$$w_i e_i(x''|_U) + w_i e_i(x^*|_R) \leq (1 + \lambda\epsilon) c_{opt},$$

where λ is a finite integer. Let x' be a concatenation of $x^*|_R$ and $x''|_U$. Then

$$w_i e_i(x') = w_i e_i(x^*|_R) + w_i e_i(x''|_U) \leq (1 + \lambda\epsilon) c_{opt}.$$

In conclusion, we have ultimately developed a PTAS for a solution of the problem. We recapitulate it in Figure 1 and conclude it with a theorem.

Theorem 3. *There is a PTAS to the MCHEWC problem.*

5 Conclusion Remarks

Our work explores provably good algorithm for the MCHEWC problem by presenting a PTAS. The aim is to extend the techniques in [7] for the MCHEC problem to a weighted version. Further challenging problems relative to this topic include whether there exists a PTAS when both the hyperedges and edges in the cycle are weighted. This most general problem remains open.

References

1. Andras, F.: Edge-disjoint paths in planar graphs. J. Combin. Theory Ser. B 38, 164–178 (1985)
2. Andras, F., Takao, N., Nobuji, S., Hitoshi, S., Eva, T.: Algorithms for routing around a rectangle. Discrete Applied Mathematics 40, 363–378 (1992)

3. Joseph, L.G., James, P.C.: Minimum-congestion hypergraph embedding in a cycle. IEEE Trans. Comput. 46(5), 600–602 (1997)
4. Qianping, G., Yong, W.: Efficient algorithm for embedding hypergraph in a cycle. IEEE Transactions on Parallel and Distributed Systems 17(3), 205–214 (2006)
5. Singling, L., Hannjang, H.: Algorithms and complexity for weighted hypergraph embedding in a cycle. In: Proc. of the 1st International Symposium on Cyber World (2002)
6. Singling, L., Hannjang, H.: A 1.5 approximation algorithm for embedding hyperedges in a cycle. IEEE Transactions on Parallel and Distributed Systems 16(6), 481–488 (2005)
7. Guojun, L., Xiaotie, D., Ying, X.: A Polynomial Time Approximation Scheme for Embedding Hypergraph in a Cycle. IEEE/ACM Transactions on Algorithms (to appear)

FPTAS's for Some Cut Problems in Weighted Trees*

Mingyu Xiao[1], Takuro Fukunaga[2], 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
{takuro,nag}@amp.i.kyoto-u.ac.jp

Abstract. Given a tree with nonnegative edge cost and nonnegative vertex weight, and a number $k \geq 0$, we consider the following four cut problems: cutting vertices of weight at most or at least k from the tree by deleting some edges such that the remaining part of the graph is still a tree and the total cost of the edges being deleted is minimized or maximized. The MinMstCut problem (cut vertices of weight *at most* k and *minimize* the total cost of the edges being deleted) can be solved in linear time and space and the other three problems are NP-hard. In this paper, we design an $O(ln/\varepsilon)$-time $O(l^2/\varepsilon + n)$-space algorithm for MaxMstCut, and $O(ln(1/\varepsilon + \log n))$-time $O(l^2/\varepsilon + n)$-space algorithms for MinLstCut and MaxLstCut, where n is the number of vertices in the tree, l the number of leaves, and $\varepsilon > 0$ the prescribed error bound.

Keywords: Graph Cut, FPTAS, Tree, Tree Knapsack.

1 Introduction

Given an undirected tree $G = (V, E)$ with a nonnegative vertex weight $w : V \to \Re^*$ and a nonnegative edge cost $c : E \to \Re^*$, and a nonnegative number $k \geq 0$, the *maximum cutting vertices of weight at most* k problem (MaxMstCut) is to find a nonempty and proper subset X of V that *maximizes* the total cost of edges with exact one endpoint in X under the constraint that (i) $G[V - X]$ is connected and (ii) the total weight of vertices in X is *at most* k. Analogously, we also define MaxLstCut, MinMstCut and MinLstCut, by replacing "maximizes" with "minimizes" or replacing "at most" with "at least" in the definition. If G is a rooted tree, then we impose the third constraint that the root is contained in the subgraph $G[V-X]$ in finding a solution X. In this paper, our algorithms may

* This work was partially supported by Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan, and National Natural Science Foundation of China under the Grant 60903007. Part of the work was done when the first author was visiting Nagamochi's lab in Kyoto University.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 210–221, 2010.

compute a solution $X \subset V$ by identifying the corresponding set $C = E(X, V-X)$ of edges, called *cut set*.

Graph cut problems have great applications and have been extensively studied in the literature. The optimally cutting k vertices (or vertices of weight k) problem in general graphs is a hard problem. Few 'positive' results on the problem are known. Considering the hardness of the problem, we study the problem in the graphs restricted to trees. The different versions of the problem in trees are still interesting, since they naturally generate the famous *knapsack problem* from several ways [8].

Our problems in rooted trees are related to some special cases of the *precedence constraint knapsack problem* (PCKP) [5,3]. In PCKP, a directed acyclic graph is given, each vertex in the graph is regarded as an item to be selected into the knapsack, and if a vertex is selected into the knapsack then all its *ancestors* (vertices having a direct path to the vertex) must also be selected into the knapsack. MaxMstCut (resp. MinLstCut) can model the precedence constraint knapsack problem in rooted trees with all edges directed toward (resp. out) the root, which is called the *in-tree* (resp. *out-tree*) *knapsack problems* [3,1].

As a prior work on MinLstCut in rooted trees, there exists a $(4/3 + \epsilon)$-approximation algorithm for the partial totally unimodular cover problem with any $\epsilon > 0$, presented by Könemann, Parekh and Segev [6]. MinLstCut is contained by their problem. Another related work is the algorithms for the tree knapsack problems. Cho and Shaw presented a depth-first dynamic programming algorithm for the out-tree knapsack problem in [1] that can be modified to get an $O(n(W-k))$-time pseudo polynomial-time algorithm for MinLstCut in rooted trees, where W is the total weight of all vertices in the tree. This pseudo polynomial-time algorithm can also be used to derive approximation schemes, but the running time depends on W. Johnson and Niemi [3] designed several algorithms for two different tree knapsack problems. Their $O(n^3/\varepsilon)$-time FPTAS (fully polynomial-time approximation scheme) for the in-tree knapsack problem can solve MaxMstCut in rooted trees in the same time. There are few nontrivial results on the problems in unrooted trees. In this paper, except showing that MinMstCut in trees can be solved in linear time and space easily, we present:

(i) $O(ln/\varepsilon)$-time $O(l^2/\varepsilon + n)$-space FPTAS's for MaxMstCut in rooted and unrooted trees;

(ii) $O(ln(1/\varepsilon + \log n))$-time $O(l^2/\varepsilon + n)$-space FPTAS's for MinLstCut and MaxLstCut in rooted and unrooted trees.

The rest of the paper is organized as follows. Section 2 gives our notation system. Section 3 presents pseudo polynomial-time algorithms that will be used to design FPTAS's. Section 4 presents FPTAS for MaxMstCut in rooted trees and Section 5 presents FPTAS's for MinLstCut and MaxLstCut in rooted trees. Finally Section 6 designs FPTAS's for the problems in unrooted trees.

2 Preliminaries

For an undirected graph $G = (V, E)$ and a subset $X \subseteq V$, let $G[X]$ denote the subgraph of G induced by the vertices in X, and let $E(X, V - X)$ denote the

set of edges with one endpoint in X and the other in $V - X$. We may denote by $V(G)$ and $E(G)$ the sets of vertices and edges of a graph G, respectively. Let $\Re^* = \Re^+ \cup \{0\}$ denote the set of nonnegative reals. For a vertex weight $w : V \to \Re^*$ and an edge cost $c : E \to \Re^*$, let $w(X)$ denote $\sum_{v \in X} w(v)$ and $c(X)$ denote $\sum_{e \in E(X, V-X)} c(e)$, respectively. Given an unrooted tree G and a vertex r in it, we use G_r to denote the rooted tree obtained from G by choosing r as the root. In a rooted tree, let $V(v)$ denote the set of descendants of a vertex v, where $v \in V(v)$, and $e_p(v)$ denote the edge connecting v and its parent. A set X is called *root-connecting* if $G[V - X]$ is connected and contains the root. We use n and l to denote the number of vertices and the number of leaves (i.e., vertices of degree 1) in the graph respectively. To distinguish edge weight with vertex weight, we say edge cost instead of edge weight.

We will consider the four problems MaxMstCut, MaxLstCut, MinMstCut and MinLstCut in rooted and unrooted trees with nonnegative edge cost and nonnegative vertex weight. MaxMstCut, MaxLstCut and MinLstCut are NP-hard as they generate the well-known knapsack problem (more discussions can be found in the full version of this paper [8]). In the full version, we also show that these three problems in trees are polynomial-time solvable when the vertex weight can only be 0 or 1. However, MinMstCut is polynomial-time solvable, even without the constraint on the vertex weight. Note that the edge cost and vertex weight are nonnegative. The cut set corresponding to the optimal solution only contains one edge. We only need to identify a *feasible* edge with minimum cost. An edge e in a rooted tree is a feasible edge if $w(e) \leq k$, and an edge e in an unrooted tree is a feasible edge if at least one component of $G - e$ is of total vertex weight $\leq k$. We can find all feasible edges in a DFS (For an unrooted tree, we arbitrarily select a vertex as the root and consider if $\min(w(v), W - w(v)) \leq k$ for each vertex v). In the DFS, we only need to remember the feasible edge with minimum cost among all visited edges. Therefore, we can solve the problem in linear time and space. We will not discuss MinMstCut in the rest of the paper.

3 Pseudo Polynomial-Time Algorithms

In this section, we give dynamic programming algorithms for our problems in rooted trees, which are pseudo polynomial-time algorithms and will be used to derive FPTAS's. One technique we used to design the dynamic programming algorithms is called the 'left-right' method for trees in [3]. To get the pseudo polynomial-time algorithms, we assume that all edge cost are integers in this section. In fact, our algorithm will find an optimal solution X only when $c(X) \leq q$ holds for a prescribed threshold q.

We order all the leaves in the tree according to a DFS (first visited has small index). In our algorithm, if a vertex is not allowed to be cut away from the rooted tree, then all the edges in the path from the vertex to the root are not allowed to be deleted. Let $L = \{z_1, z_2, \ldots, z_l\}$, $l = |L|$, be the set of leaves, where leaves $z_1, z_2, \ldots, z_l$ appear from left to right in this order. For $0 \leq i \leq l$ and $0 \leq j \leq q$, define $\mathcal{X}(i, j)$ to be the set of all root-connecting sets $X \subseteq V - \{r\}$ that satisfy

two constraints: $L \cap X \subseteq \{z_1, z_2, \ldots, z_i\}$ (only the first i leaves are allowed to be cut away); and $c(X) = j$.

Let $OPT(i,j) = \min\{w(X) \mid X \in \mathcal{X}(i,j)\}$ for MaxMstCut (resp., $OPT(i,j) = \max\{w(X) \mid X \in \mathcal{X}(i,j)\}$ for MaxLstCut and MinLstCut), and let $X(i,j)$ store a set $X \in \mathcal{X}(i,j)$ attaining $w(X) = OPT(i,j)$, where we let $X(i,j) = \perp$ and $OPT(i,j) = \infty$ if $\mathcal{X}(i,j) = \emptyset$. Clearly $X(l, j_0)$ is an optimal solution to MaxMstCut if j_0 is the maximum number such that $OPT(l, j_0) \leq k$. We maintain a table of size $(l+1) \times (q+1)$ to store $OPT(i,j)$ and $X(i,j)$ for $0 \leq i \leq l$ and $0 \leq j \leq q$. Let P_i be the set of edges such that we can remove z_i without deleting z_{i+1} by making a cut at any edge from P_i. Then, for any edge $e \in P_i$, deleting e cuts away leaves $z_a, z_{a+1}, \ldots, z_i$ with consecutive indices. Let $\alpha(e)$ be the smallest index a of the leaves being cut. Then all values $OPT(i,j)$ and $X(i,j)$ can be computed by the recursive formula in Fig. 1.

Algorithm DP(G, k, q)
Input: A tree $G = (V, E)$ rooted at a vertex r with nonnegative integer edge cost and nonnegative vertex weight and two numbers $k, q \geq 0$.
Output: A solution to MaxMstCut in G if the value of the optimal solution is at most q.

1. For $j = 1$ to q, do
   $OPT(0,j) \longleftarrow \infty$, $X(0,j) \longleftarrow \perp$.
2. For $i = 1$ to l, do
   $OPT(i,0) \longleftarrow 0$, $X(i,0) \longleftarrow \emptyset$.
   For $j = 1$ to q, do
   $OPT(i,j) \longleftarrow OPT(i-1,j)$, $X(i,j) \longleftarrow X(i-1,j)$.
   For each $e \in P_i$, do
   If $c(e) \leq j$ and $OPT(\alpha(e)-1, j-c(e)) + w(e) < OPT(i,j)$, then
   $OPT(i,j) \longleftarrow OPT(\alpha(e)-1, j-c(e)) + w(e)$,
   $X(i,j) \longleftarrow X(\alpha(e)-1, j-c(e)) \cup V(e)$.
3. Let j_0 be the maximum number such that $OPT(l, j_0) \leq k$.
4. Return $X(l, j_0)$.

Fig. 1. Algorithm DP(G, k, q)

Lemma 1. *For a rooted tree G with nonnegative integer edge cost and nonnegative vertex weight, Algorithm* DP(G, k, q) *can be implemented to run in $O(qn)$ time and $O(ql + n)$ space.*

Note that $\{P_i \mid i = 1, 2, \ldots, l\}$ is a partition of the edge set E. So each edge is considered only once in the algorithm. Then the running time of the algorithm is $O(qn)$. The space of the algorithm is $O(ql + n)$.

Since DP(G, k, q) only computes $OPT(i,j)$ for j from 0 to q, when the weight of an optimal solution is greater than q, DP(G, k, q) can not find an optimal solution.

4 FPTAS for MaxMstCut in Rooted Trees

First, we consider MaxMstCut in rooted trees and then extend our results to the problem in unrooted trees in Section 6.

4.1 A $\frac{1}{3}$-Approximation Algorithm

To design our FPTAS for MaxMstCut, we first give an $O(n \log l)$-time $\frac{1}{3}$-approximation algorithm. The idea of the algorithm comes from a well-known greedy approximation algorithm for the knapsack problem, which is to greedily select items of the most *dense*. Let V' be the set of all vertices except the root. For any $v \in V'$, the *density* of v is defined as $\varphi(v) = c(e)/w(V(v))$, where $e = e_p(v)$. Our algorithm will first contract all *overload* and *dominated edges* as preprocesses. An edge is called an *overload edge*, if the total weight of vertices cut away from the tree by deleting it exceeds k. Clearly, we can simply contract all overload edges to form the new root. An edge e is called a *dominated edge*, if we can cut away an edge e' with cost $c(e') > c(e)$ by deleting e. It is also easy to see that no dominated edge will be in the cut set corresponding to an optimal solution. We can also safely contract all dominated edges directly, and in the new tree, on any path from the root to a leaf, the edge cost is monotonous. The simple approximation algorithm is presented in Fig. 2.

Algorithm APPRO(G, k)
Input: A rooted tree $G = (V, E)$ with nonnegative edge cost $c : E \to \Re^*$ and nonnegative vertex weight $w : V \to \Re^*$ and a number $k > 0$.
Output: A $\frac{1}{3}$-approximation solution to MaxMstCut in G.

1. Contract all overload and dominated edges.
2. $A \longleftarrow$ the set of vertices v with $w(v) = 0$.
3. While $V' - A \neq \emptyset$, do select $v \in V' - A$ such that $\varphi(v)$ is maximized
   If $w(A) + w(V(v)) \leq k$, add $V(v)$ into A.
   Else return the better of A and $V(v)$ and halt.
4. Return A.

Fig. 2. Algorithm APPRO(G, k)

Lemma 2. *Algorithm* APPRO(G, k) *is an* $O(n \log l)$*-time* $\frac{1}{3}$*-approximation algorithm for* MaxMstCut *in rooted trees with nonnegative edge cost and nonnegative vertex weight.*

Proof. Since we have contracted all dominated edges, we know that there is an optimal solution contains all vertices with zero weight in the tree. Then we can simply put them into the solution set directly. Next, if the algorithm stops at Step 4, it will return an optimal solution, else the algorithm will return A or $V(v')$ in Step 3, where v' is the last vertex being selected in the algorithm. Note that

$V(v')$ is excluded from A in the algorithm. Thus A is a feasible solution when the algorithm halts. $V(v')$ is also a feasible solution, because all overload edges are contracted in Step 1. Assume that X^* is an optimal solution. We partition A and X^* into a serial of disjoint subsets: $A = \{V(v_1), V(v_2), \ldots, V(v_x)\}$ and $X^* = \{V(u_1), V(u_2), \ldots, V(u_y)\}$, where v_i is not a descendent of any vertex in A and u_i is not a descendent of any vertex in X^*. We further assume that $\{u_1, \ldots, u_j\} \subseteq A$ and $\{u_{j+1}, \ldots, u_y\} \cap A = \emptyset$. Obviously, $X_1 = \{V(u_1), V(u_2), \ldots, V(u_j)\}$ is still a feasible solution and $X_1 \subseteq A$. Since we have contracted all dominated edges, we know that $c(A) \geq c(X_1)$. If $c(X_1) \geq \frac{1}{3}c(X^*)$, then $c(A) \geq \frac{1}{3}c(X^*)$. Otherwise, we have $c(X_2) \geq \frac{2}{3}c(X^*)$, where $X_2 = X^* - X_1 = \{V(u_{j+1}), \ldots, V(u_y)\}$. Since at the time when the algorithm selects v', no vertex in $\{u_{j+1}, \ldots, u_y\}$ has been selected by our algorithm, we know that $\varphi(v_i) \geq \varphi(v') \geq \varphi(u_{i'})$ holds for any $i = 1, \ldots, x$ and $i' = j+1, \ldots, y$, which implies the total cost of A and $V(v')$ is greater than that of X_2. Then $c(A) + c(V(v')) \geq c(X_2) \geq \frac{2}{3}c(X^*)$. Therefore, in any case, the better of A and $V(v')$ will has cost at least $\frac{1}{3}c(X^*)$.

As for the running time, we only need to show that Step 3 can be done in $O(n \log l)$ time. First of all, we compute the densities of all vertices in the tree in a DFS. If a vertex v has an ancestor u such that $\varphi(v) < \varphi(u)$, we say that v is a *weak vertex* and $e_p(v)$ a *weak edge*. Note that the algorithm will never select a weak vertex in Step 3. We can reduce the tree by contracting all weak edges. Detecting and contracting all weak edges can be done in a DFS. After reducing the tree, the tree has the *monotonicity property*: on any path from the root to a leaf, the vertex density is monotonous. Then we can sort all the vertices according to their densities in $O(n \log l)$ time as we do in Mergesort. After sorting the vertices, we only need to select vertices according to the sorting list. Then we can do Step 3 in $O(n \log l)$ time. ∎

Note. This greedy algorithm can get approximation ratio $\frac{1}{2}$ for the knapsack problem. Ibarra and Kim [2] also introduced an $O(n^2)$-time $\frac{1}{2}$-approximation algorithm for MaxMstCut in rooted trees. In their algorithm, they will update the densities of vertices when each time selecting a vertex of maximum density and deleting it from the tree[1]. We will use our $\frac{1}{3}$-approximation algorithm instead of their $\frac{1}{2}$-approximation algorithm, because we cannot improve the running time bound of our FPTAS by improving the approximation ratio from $\frac{1}{3}$ to $\frac{1}{2}$, but will worsen the running time bound if the running time of the constant approximation algorithm changes from $O(n \log l)$ to $O(n^2)$.

4.2 The FPTAS

Next, we design the FPTAS. Let $R = \textsc{Appro}(G, k)$. Algorithm $\mathrm{DP}(G, k, 3c(R))$ will find an optimal solution for our problem (assume that all edge costs are integers). If the cost of each edge is a small number, i.e., bounded by a polynomial of n, then $\mathrm{DP}(G, k, 3c(R))$ runs in polynomial time. To obtain an FPTAS, we

[1] The correctness of the algorithm (Theorem 2 in [2]) was proved by assuming an important property (the fourth line of the proof of Theorem 2), the complete proof of which was omitted.

scale the cost of each edge down to be bounded by a polynomial of n and use Algorithm DP on the new instance to get a solution. By scaling with respect to the desired error ε, we can get a solution with cost within $(1-\varepsilon)c(X^*)$ and $c(X^*)$ in polynomial time with respect to both n and $1/\varepsilon$, where X^* is an optimal solution to our problem. The FPTAS for MaxMstCut is described in Fig. 3.

Algorithm FPTAS1(G, k, ε)
Input: A rooted tree $G = (V, E)$ with nonnegative edge cost $c : E \to \Re^*$ and nonnegative vertex weight $w : V \to \Re^*$, and two numbers $k \geq 0$ and $1 > \varepsilon \geq 0$.
Output: An $(1-\varepsilon)$-approximation solution to MaxMstCut.

1. Apply preprocesses to deal with overload edges and idle edges.
2. $R \longleftarrow$ APPRO(G, k) and $g \longleftarrow c(R)$.
3. If $g = 0$, return R as an optimal solution and halt.
4. $Q \longleftarrow \varepsilon g/l$ and replace $c(e)$ with $c'(e) = \lfloor c(e)/Q \rfloor$.
5. Return DP$(G, k, \lfloor 3g \rfloor)$.

Fig. 3. Algorithm FPTAS1(G, k, ε)

Theorem 1. *For* MaxMstCut *in a rooted tree with nonnegative edge cost and nonnegative vertex weight, algorithm* FPTAS1(G, k, ε) *runs in* $O(ln/\varepsilon)$ *time and* $O(l^2/\varepsilon + n)$ *space, and returns a set* A *such that*

$$c(A) \geq (1-\varepsilon)c(X^*), \tag{1}$$

where X^* *is an optimal solution to* MaxMstCut.

Proof. Since APPRO(G, k) runs in $O(n \log l)$ time and $O(n)$ space, and DP $(G, k, \lfloor 3g \rfloor)$ runs in $O(3gn/Q) = O(ln/\varepsilon)$ time and $O(3gl/Q + n) = O(l^2/\varepsilon + n)$ space, we get the running time and space bounds.

Next, we prove (1). Since the cardinality of any cut is at most the number l of leaves in G, we have that $|E(V - X^*, X^*)| \leq l$. It is also clear that for any $e \in E(V - X^*, X^*)$, $0 \leq c(e) - Qc'(e) < Q$. Then we have

$$0 \leq c(X^*) - Qc'(X^*) < lQ.$$

Since A is an optimal solution for the scaled instance, it must be at least as good as X^* in the scaled instance. Then

$$\begin{aligned} c(A) &\geq Qc'(X^*) \\ &\geq c(X^*) - lQ = c(X^*) - \varepsilon g \quad (\text{by } c(X^*) \geq g) \\ &\geq (1-\varepsilon)c(X^*). \end{aligned}$$

∎

5 FPTAS's for MinLstCut and MaxLstCut in Rooted Trees

We cannot directly extend the FPTAS for MaxMstCut in Section 4 to an FPTAS for MinLstCut or MaxLstCut in rooted trees, because we have no similar constant factor algorithm as APPRO(G, k) for MinLstCut or MaxLstCut. To get a

constant factor algorithm, we will use a 'rounding' technique [7]. This technique is used as a basic trick for deriving FPTAS's for many problems. We will apply it in a straightforward way to our algorithm to get an approximation scheme first. Then we further improve the running time of our algorithm by using the 'rounding' technique again to get the final FPTAS. In this section, we design the FPTAS for MinLstCut, which can also be modified to solve MaxLstCut.

5.1 The Preliminary Algorithm

Firs, we present an algorithm SCALING that, given a MinLstCut instance $I = (G, c, w, k)$, a desired error ratio ε and a number p, will run in $O(ln/\varepsilon)$ time and $O(l^2/\varepsilon + n)$ space and return A, where $A = \perp$, if $c(X^*) > 2p$, and A is a feasible solution with value $c(A)$ satisfying $(1+\varepsilon)c(X^*) \geq c(A)$, if $p \leq c(X^*) \leq 2p$.

Algorithm SCALING(G, k, ε, p)
Input: A rooted tree $G = (V, E)$ with nonnegative edge cost $c : E \to R^*$ and nonnegative vertex weight $w : V \to R^*$, and three numbers $k \geq 0$, $1 > \varepsilon \geq 0$ and $p \in N$.
Output: $\perp$ or a feasible solution.

1. $Q \longleftarrow \varepsilon p/l$ and replace $c(e)$ with $c'(e) = \lfloor c(e)/Q \rfloor$.
2. Return DP$(G, k, 2p)$.

Fig. 4. Algorithm SCALING(G, k, ε, p)

By the definition of DP$(G, k, 2p)$ for MinLstCut, we know that if $c(X^*) > 2p$, our algorithm will return $\perp$, else will return a feasible solution A such that $c(A) \geq c(X^*)$. Assume $p \leq c(X^*) \leq 2p$ and let $A =$ SCALING(G, k, ε, p). Analogously with the analysis of Theorem 1, we can prove that directly

$$\begin{aligned} c(A) &\leq Q \cdot c'(X^*) \\ &\leq c(X^*) + lQ = c(X^*) + \varepsilon p \quad (\text{by } c(X^*) \geq p) \\ &\leq (1+\varepsilon)c(X^*). \end{aligned}$$

Therefore, SCALING(G, k, ε, p) works as claimed.

To get an FPTAS, we need to compute a value p^* such that $p^* \leq c(X^*) \leq 2p^*$ and call SCALING(G, k, ε, p^*). We may simply assume $c(X^*) \geq 1$. Then we perform the loop in Fig. 5 to compute p^*.

It is clear that the process in Fig. 5 will return p^* by calling SCALING$(G, k, \varepsilon = 1, p)$ for at most $\lceil \log c(X^*) \rceil$ times. Therefore, we can compute p^* in $O(ln \log c(X^*))$ time and $O(l^2 + n)$ space.

Theorem 2. *There is an $O(ln(1/\varepsilon + \log c(X^*))$-time and $O(l^2/\varepsilon + n)$-space algorithm that computes a solution with error ε for* MinLstCut *in a rooted tree with nonnegative edge cost and nonnegative vertex weight.*

1. Let $p \longleftarrow 1$;
2. If SCALING$(G, k, \varepsilon = 1, p) \neq \perp$, return p as p^* and halt;
3. Otherwise, let $p \longleftarrow 2p$ and go to Step 2.

Fig. 5. Computing p^*

Since the running time of the approximation algorithm depending on $c(X^*)$, it is not a real polynomial-time algorithm. Next, we further improve the running time bound from $O(ln(1/\varepsilon + \log c(X^*)))$ to $O(ln(1/\varepsilon + \log n))$.

5.2 Further Improvement

In Fig. 5, we set p as 1 in the beginning to compute p^*, and the algorithm may run $\lceil \log c(X^*) \rceil$ loops. If we get a value p_1 such that $p_1 \leq c(X^*) \leq np_1$ and set p as p_1 in the first step, then we can compute P^* in $O(ln \log n)$ time. We show that we can also use the rounding technique to compute p_1 in $O(ln \log n)$ time.

We sort the edges according to their costs in $O(n \log n)$ time and assume that $c(e_{i_1}) \leq c(e_{i_2})$ if $i_1 \leq i_2$. Note that all edges are of positive cost edges. If $c(e_i) \leq c(X^*) \leq c(e_{i+1})$, we can contract all edges with cost greater than $c(e_i)$ safely. And in the remaining tree, the total cost of all edges is not greater than $nc(e_i)$. Therefore, $c(e_i)$ is a satisfied value of p_1. We can also use Algorithm SCALING with $\varepsilon = 1$ to detect e_i. If we check each edge, then we may call SCALING for n times. But we can simply improve it to $\lceil \log n \rceil$ times by using binary search. Then, we can compute p_1 in $O(ln \log n)$ time and $O(l^2 + n)$ space.

The FPTAS for MinLstCut in this section can be directly modified to a similar FPTAS for MaxLstCut, which preserves the approximation ratio.

Theorem 3. *There is an $O(ln(1/\varepsilon + \log n)$-time and $O(l^2/\varepsilon + n)$-space algorithm that computes a solution with error ε for* MinLstCut *and* MaxLstCut *in a rooted tree with nonnegative edge cost and nonnegative vertex weight.*

6 FPTAS's for Unrooted Trees

If the tree is an unrooted tree, we still can solve the problem by choosing each vertex as a root and solving the n problems in the rooted trees. But the running time bound will increase by a factor of $\Theta(n)$. In this section, we show that our algorithms for rooted trees can be extended to that for unrooted trees without increasing the running time bound. We will iteratively choose a *leaf-balanced separator* of the tree and solve the problem in a divide-and-conquer way.

First, we define a constrained version of our problems. Given a set of inputs of MaxMstCut additionally with a prescribed set L' of leaves, the *constrained* MaxMstCut requires to find a solution to cut away all leaves in L' (possibly

with other leaves) from the given tree, where L' may be empty. We design an algorithm DP2(G, k, L', q) for the constrained MaxMstCut. Thus for $L' = \emptyset$, DP2$(G, k, \emptyset, q)$ is exactly DP(G, k, q).

DP2(G, k, L', q) can be obtained by modifying DP(G, k, q) as follows. In the fourth line of Step 2, DP(G, k, q) initializes $OPT(i, j)$ and $X(i, j)$ by

$$OPT(i, j) \longleftarrow OPT(i-1, j),\ X(i, j) \longleftarrow X(i-1, j).$$

In the modified algorithm, the same initialization is used if $z_i \notin L'$. On the other hand, if $z_i \in L'$, we initialize them by

$$OPT(i, j) \longleftarrow \infty,\ X(i, j) \longleftarrow \perp .$$

By this modification, all cuts that keep $z_i \in L'$ in the tree are assigned an infinite cost. Thus the algorithm computes a minimum cost solution of the constrained MaxMstCut, if a feasible solution exists. Note that the running time bound of DP2(G, k, L', q) does not change.

Next, we introduce the concept of *leaf centroid*, which will be used to split the tree into several small trees.

Lemma 3. *For any tree, there is a vertex such that each of the subtrees obtained by removing it contains at most half of number of leaves in the original tree, and this vertex can be found in linear time.*

We will call the vertex described in Lemma 3 a *leaf centroid*. Leaf centroid is an extension of *centroid* introduced in [4]. Lemma 3 can be proved easily.

Recall that G_r is the rooted tree obtained from an unrooted tree G by choosing r as the root. For a neighbor v of r, let $G_r(v)$ denote the tree obtained by contracting $V - V(v)$ in G_r into a single vertex r. Let $w_r(v)$ denote the total weight of vertices in $V - V(v)$. Then $w_r(v)$ can be computed in a DFS for all pairs of r and its neighbor v.

Now we are ready to give the main steps of our algorithm for the problem in unrooted trees. Given an unrooted tree G, first, we select a leaf centroid v of the tree as a root to get a rooted tree G_r and use the FPTAS in Section 4 for MaxMstCut (or the FPTAS in Section 5 for MinLstCut) to get a candidate solution. Second, for each neighbor v of r, we consider the new created subtree $G_r(v)$ as a constrained problem by adding the new created vertex r in to L'. With this method, we can solve our problems in unrooted trees iteratively. Let X be an optimal solution, if $r \in V - X$ then we can find a solution within the error bound in the first step, if $r \in X$ then the optimal solution is an optimal solution to the problem in $G_r(v)$ for some neighbor v of r with the constraint $r \in X$, which implies that we can solve the problem by solving the problems in $G_r(v)$. We will assume that we solve the problem directly if the tree has only two leaves. Then the algorithm will run at most $O(\log l)$ iterations. Note that when the tree has at least three leaves, no leaf centroid is a leaf of the tree, and hence L' does not contain a possible root, implying that DP2(G_r, k, L', q) receives a well-defined input in our algorithm.

As for the running time, we first show that when the graph is a path (a tree has only two leaves), our problems can be solved in linear time. Clearly, it holds for MinLstCut and MaxLstCut. For MaxMstCut in unrooted trees, it is easy to see that, for each edge of maximum cost, there is an optimal solution containing at most one endpoint of it. Then we can select an edge (u, v) of maximum weight, and return the better one between two solutions in two rooted trees G_u and G_v. Therefore, the problem in unrooted paths can be solved in linear time.

The hard part is the analysis for the general case. Since the case for MinLstCut is more complicated than the case for MinLstCut. We will analyze the algorithm for MinLstCut. We consider the search tree $\mathcal{T}$ generated in the algorithm. Each node of $\mathcal{T}$ corresponds to an instance of the constrained MinLstCut in unrooted trees. The root of $\mathcal{T}$ represents the original instance, say $G^{(0)}$. The children of a node x in $\mathcal{T}$ correspond to the instances called recursively by the second step of the algorithm when it solves the instance corresponding to x. We say that a node is in level i if the path from the node to the root $G^{(0)}$ contains i edges.

Let $G^{(i-1)}$ be the tree corresponding to an instance $I^{(i-1)}$ in the $(i-1)$-st level of $\mathcal{T}$, and $G_j^{(i)}$ be the trees corresponding to the children instances $I_j^{(i)}$ of $I^{(i-1)}$ $(j = 1, 2, \ldots, x)$. We assume that $G^{(i-1)}$ has $n^{(i-1)}$ vertices and $l^{(i-1)}$ leaves, and $G_j^{(i)}$ has $n_j^{(i)}$ vertices and $l_j^{(i)}$ leaves. Then $3 \le n_j^{(i)} \le n^{(i-1)} - 1$, $3 \le l_j^{(i)} \le \lceil \frac{l^{(i-1)}}{2} \rceil + 1$, $\sum_{j=1}^{x} n_j^{(i)} \le n^{(i-1)} + x - 1$, and $\sum_{j=1}^{x} l_j^{(i)} \le l^{(i-1)} + x$. Under these constraints, we have the following relation (see full version of the paper [8] for the proof)

$$\sum_{j=1}^{x} l_j^{(i)} n_j^{(i)} \le \frac{l^{(i-1)} n^{(i-1)}}{2} + \frac{1}{2} l^{(i-1)} + 2n^{(i-1)} - 4. \tag{2}$$

Now we analyze the whole running time of the algorithm for MinLstCut. In each node of $\mathcal{T}$, the algorithm will call the FPTAS in Section 5 (say FPTAS2 for convenience) for once. The whole running time of the algorithm will be the sum of all the running time taking by FPTAS2. We will use $N^{(i)}$ to denote the total number of vertices of all trees in level i of $\mathcal{T}$. It is easy to verify that $N^{(i)} \le 3n^{(0)}$.

For instance $I^{(i-1)}$, computing FPTAS2 will use $O(l^{(i-1)} n^{(i-1)} (1/\varepsilon + \log n^{(i-1)}))$ time. For all subinstances of $I^{(i-1)}$, computing FPTAS2 will take basic computations of

$$\begin{aligned} & \textstyle\sum_{j=1}^{x} l_j^{(i)} n_j^{(i)} (1/\varepsilon + \log n_j^{(i)}) \\ \le\ & (\tfrac{l^{(i-1)} n^{(i-1)}}{2} + \tfrac{1}{2} l^{(i-1)} + 2n^{(i-1)})(1/\varepsilon + \log n^{(i-1)}) \qquad \text{(by (2))} \\ \le\ & (\tfrac{l^{(i-1)} n^{(i-1)}}{2} + \tfrac{5}{2} n^{(i-1)})(1/\varepsilon + \log n^{(i-1)}). \end{aligned}$$

This implies that in the i-th level of $\mathcal{T}$, totally we use no more than

$$(\sum_{j} \frac{l_j^{(i-1)} n_j^{(i-1)}}{2} + \frac{5}{2} N^{(i-1)}) \cdot (1/\varepsilon + \log N^{(i-1)})$$

basic computations. By iteratively using the above relation, we get that

$$\begin{aligned}
&(\textstyle\sum_j l_j^{(i-1)} n_j^{(i-1)}) \cdot (1/\varepsilon + \log N^{(i-1)}) \\
&\le \tfrac{l^{(0)} n^{(0)}}{2^{i-1}}(1/\varepsilon + \log n^{(0)}) + \textstyle\sum_{j=0}^{i-2} \frac{5/2 \cdot N^{(j)}}{2^{(i-2-j)}}(1/\varepsilon + \log N^{(j)}) \\
&\le \tfrac{l^{(0)} n^{(0)}}{2^{i-1}}(1/\varepsilon + \log n^{(0)}) + 5N^{(i-1)}(1/\varepsilon + \log N^{(i-1)}) \\
&\le \tfrac{l^{(0)} n^{(0)}}{2^{i-1}}(1/\varepsilon + \log n^{(0)}) + 15n^{(0)}(1/\varepsilon + \log 3n^{(0)}). \qquad (\text{by } N^{(i-1)} \le 3n^{(0)})
\end{aligned}$$

Therefore, in the i-th level of $\mathcal{T}$, we use basic computations no more than

$$\begin{aligned}
&\tfrac{1}{2}(\tfrac{l^{(0)} n^{(0)}}{2^{i-1}}(1/\varepsilon + \log n^{(0)}) + 15n^{(0)}(1/\varepsilon + \log 3n^{(0)})) + \tfrac{15}{2} n^{(0)}(1/\varepsilon + \log 3n^{(0)}) \\
&\le \tfrac{l^{(0)} n^{(0)}}{2^i}(1/\varepsilon + \log n^{(0)}) + 15n^{(0)}(1/\varepsilon + \log 3n^{(0)}).
\end{aligned}$$

Since $\mathcal{T}$ has $O(\log l^{(0)})$ levels, totally, we use basic computations of

$$\begin{aligned}
&\textstyle\sum_i (\frac{l^{(0)} n^{(0)}}{2^i}(1/\varepsilon + \log n^{(0)}) + 15n^{(0)}(1/\varepsilon + \log 3n^{(0)})) \\
&= O(l^{(0)} n^{(0)}(1/\varepsilon + \log n^{(0)})).
\end{aligned}$$

We can solve MinLstCut in unrooted trees with the above running time bound. In the same way, we know that MaxMstCut in unrooted trees can also be solved in the same running time as that for the problem in rooted trees. The space used in the algorithms for the problems in unrooted trees also do not increase. Then we get

Theorem 4. *There is an $O(ln/\varepsilon)$-time and $O(l^2/\varepsilon + n)$-space algorithm that computes a solution with error ε for* MaxMstCut *in an unrooted tree with nonnegative edge cost and nonnegative vertex weight.*

Theorem 5. *There is an $O(ln(1/\varepsilon + \log n))$-time and $O(l^2/\varepsilon + n)$-space algorithm that computes a solution with error ε for* MinLstCut *and* MaxLstCut *in an unrooted tree with nonnegative edge cost and nonnegative vertex weight.*

References

1. Cho, G., Shaw, D.X.: A Depth-First Dynamic Programming Algorithm for the Tree Knapsack Problem. INFORMS Journal on Computing 9(4), 431–438 (1997)
2. Ibarra, O., Kim, C.: Approximation algorithms for certain scheduling problems. Math. Oper. Res. 3(3), 197–204 (1978)
3. Johnson, D.S., Niemi, K.A.: On Knapsacks, Partitions, and a New Dynamic Programming Technique for Trees. Mathematics of Operations Research 8(1), 1–14 (1983)
4. Jordan, C.: Sur Les Assemblages De Lignes. J. Reine Angew. Math. 70, 185–190 (1869)
5. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Heidelberg (2004)
6. Könemann, J., Parekh, O., Segev, D.: A unified approach to approximating partial covering problems. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 468–479. Springer, Heidelberg (2006)
7. Sahni, S.: General Techniques for Combinatorial Approximation. Operations Research 25(6), 920–936 (1977)
8. Xiao, M., Fukunaga, T., Nagamochi, H.: Optimally Cutting a Prescribed Number of Vertices Away From a Tree (manuscript) (2010)

Deterministic Online Call Control in Cellular Networks and Triangle-Free Cellular Networks

Joseph Wun-Tat Chan[1], Francis Y.L. Chin[2,⋆], Xin Han[3,⋆⋆], Ka-Cheong Lam[2,4], Hing-Fung Ting[2,⋆⋆⋆], and Yong Zhang[2]

[1] College of International Education, Hong Kong Baptist University, Hong Kong
cswtchan@gmail.com
[2] Department of Computer Science, The University of Hong Kong, Hong Kong
{chin,hfting,yzhang}@cs.hku.hk
[3] School of Software, Dalian University of Technology, China
hanxin.mail@gmail.com
[4] College of Computer Science, Zhejiang University, China
pandaman@163.com

Abstract. Wireless Communication Networks based on Frequency Division Multiplexing (FDM in short) plays an important role in the field of communications, in which each request can be satisfied by assigning a frequency. To avoid interference, each assigned frequency must be different to the neighboring assigned frequencies. Since frequency is a scarce resource, the main problem in wireless networks is how to utilize the frequency as fully as possible. In this paper, we consider the call control problem. Given a fixed bandwidth of frequencies and a sequence of communication requests, in handling each request, we must immediately choose an available frequency to accept (or reject) it. The objective of call control problem is to maximize the number of accepted requests. We study the asymptotic performance, i.e., the number of requests in the sequence and the number of available frequencies are very large positive integers. In this paper, we give a 7/3-competitive algorithm for call control problem in cellular network, improving the previous 2.5-competitive result. Moreover, we investigate the triangle-free cellular network, propose a 9/4-competitive algorithm and prove that the lower bound of competitive ratio is at least 5/3.

1 Introduction

Frequency Division Multiplexing (FDM in short) is commonly used in wireless communications. To implement FDM, the wireless network is partitioned into small regions (cell) and each cell is equipped with a base station. When a call request arrives at a cell, the base station in this cell will assign a frequency to

⋆ Research supported by HK RGC grant HKU-7113/07E and the William M.W. Mong Engineering Research Fund.
⋆⋆ Partially supported by Start-up Funding (1600-893335) provided by DUT, China.
⋆⋆⋆ Research supported by HK RGC grant HKU-7171/08E.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 222–233, 2010.

this request, and the call is established via this frequency. Since frequency is a scarce resource, to satisfy the requests from many users, we have to reuse the same frequency for different call requests. But if two neighboring calls are using the same frequency, interference will appear to violate the quality of communications. Thus, to avoid interference, the same frequency cannot be assigned to two different calls with distance close to each other. In general, the same frequency cannot be assigned to two calls in the same cell or neighboring cells.

There are two research directions on the fully utilization of the frequencies. One is *frequency assignment problem*, and the other is *call control problem*. In frequency assignment problem, each call request must be accepted, and the objective is to minimize the number of frequencies to satisfy all requests. In call control problem, the bandwidth of frequency is fixed, thus, when the number of call requests in a cell or in some neighboring cells is larger than the total bandwidth, the request sequence cannot be totally accepted, i.e., some requests would be rejected. The objective of call control problem is to accept the requests as many as possible.

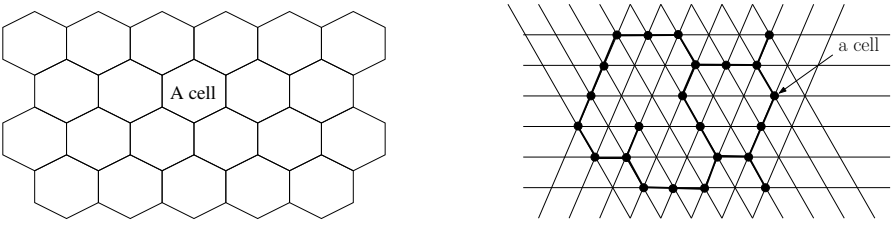
Problem Statement

In this paper, we consider the online version of call control problem. There are ω frequencies available in the wireless networks. A sequence σ of call requests arrives over time, where $\sigma = \{r_1, r_2, ..., r_t, ...\}$, c_t denote the t-th call request and also represent the cell where the t-th request arrives. When a request arrives at a cell, the system must either choose a frequency to satisfy this request without interference with other assigned frequencies in this cell and its neighboring cells, or reject this request. When handling a request, the system does not know any information about future call requests. We assume that when a frequency is assigned to a call, this call will never terminate and the frequency cannot be changed. The objective of this problem is to maximize the number of accepted requests.

We focus on the call control problem in cellular networks and triangle-free cellular networks. In the cellular network, each cell is a hexagonal region and has six neighbors, as shown in Figure 1(a). The cellular network is widely used in wireless communication networks. A network is *triangle-free* if there are no 3-cliques in the network, i.e., there are no three mutually-adjacent cells. An example of a triangle-free cellular network is shown in Fig. 1(b).

Performance Measure

To measure the performance of online algorithms, we often use the competitive ratio, which compare the output between the online algorithm and the optimal offline algorithm, which knows the whole request sequence in advance. In call control problem, the output is the number of accepted requests. For a request sequence σ, let $A(\sigma)$ and $O(\sigma)$ denote the number of accepted request of an online algorithm A and the optimal offline algorithm O, respectively. The competitive ratio of algorithm A is $R_A = \sup_\sigma O(\sigma)/A(\sigma)$. For the call control problem, we focus on the asymptotic performance, i.e., the number of requests and the



(a) cellular network (b) triangle-free cellular network

Fig. 1. An example of the cellular network and triangle-free cellular network

number of frequencies are large positive integers. The asymptotic competitive ratio for an online algorithm A is

$$R_A^{\infty} = \limsup_{n\to\infty} \max_{\sigma} \{\frac{O(\sigma)}{A(\sigma)} | O(\sigma) = n\}.$$

Related Works

How to fully utilize the frequencies to satisfy the communication requests is a very fundamental problem in theoretical computer science and engineering. Both the frequency assignment problem and the call control problem are well studied during these years. From the description of these two problems, we know that the call control problem is the dual problem of the frequency assignment problem.

The offline version of the frequency assignment problem in cellular networks was proved to be NP-hard by McDiarmid and Reed [6], and two 4/3-approximation algorithms were given in [6,7]. For the online frequency assignment problem, when a call request arrives, the network must immediately assign a frequency to this call without any interference. There are mainly three strategies: Fixed Allocation [5], Greedy Assignment [1], and Hybrid Assignment [3]. If the duration of each call is infinity and the assigned frequency cannot be changed, the hybrid algorithm gave the best result for online frequency assignment, i.e., a 2-competitive algorithm for the absolute performance and a 1.9126-competitive algorithm for the asymptotic performance. When the background network is triangle-free, a 2-local 5/4-competitive algorithm was given in [9], an inductive proof for the 7/6 ratio was reported in [4], where k-local means when assigning a frequency, the base station only knows the information of its neighboring cells within distance k. In [11], a 1-local 4/3-competitive algorithm was given.

For the call control problem, the offline version is NP-hard too [6]. To handle such problem, greedy strategy is always the first try, when a call request arrives, the network choose the minimal available frequency to serve this request, if any frequency is interfere with some neighboring assigned frequency, the request will be rejected. Pantziou et al. [8] analyzed the performance of the greedy strategy, proved that the asymptotic competitive ratio of the greedy strategy is equal to the maximal degree of the network. Caragiannis et al. [1] gave a randomized algorithm for the call control problem in cellular networks, the asymptotic competitive ratio of their algorithm is 2.651. Later, the performance of the

randomized algorithms was improved to 16/7 by the same authors [2], they also proved the lower bound of the asymptotic competitive ratio for the randomized algorithm is at least 2. Very recently, a deterministic algorithm with asymptotic competitive ratio 2.5 was given in [10], and the lower bound of the asymptotic competitive ratio for the deterministic algorithm was proved to be 2.

Our Contributions
In this paper, we consider the deterministic algorithms for the call control problem in cellular networks and triangle-free cellular networks. In cellular network, we give a 7/3-competitive algorithm, improving the previous 2.5-competitive result. In triangle-free network, we propose a 9/4-competitive algorithm, moreover, the lower bound of the competitive ratio in triangle-free network is proved to be at least 5/3.

2 Call Control in Cellular Networks

2.1 Algorithm

The idea of our algorithm for call control problem in cellular networks is similar to the algorithm in [10]. By using a totally different analysis, we can show our algorithm is better, moreover, our algorithm is best possible among this kind of algorithms.

Cellular networks are 3 colorable, each cell can be associated with a color from $\{R, G, B\}$ and any two neighboring cells are with different colors. Partition the frequencies into four sets, F_R, F_B, F_G, and F_S, where F_X $(X \in \{R, G, B\})$ can be only used in cells with color X and F_S can be used in any cell. Since we consider the asymptotic performance of the call control problem, we may regard the number ω of frequencies in the system is a multiple of 7. Divide the the frequencies into four disjoint set as follows:

$$
\begin{aligned}
F_R &= \{1, ..., 2\omega/7\}, \\
F_G &= \{2\omega/7 + 1, ..., 4\omega/7\}, \\
F_B &= \{4\omega/7 + 1, ..., 6\omega/7\}, and \\
F_S &= \{6\omega/7 + 1, ..., \omega\}
\end{aligned}
$$

Obviously, the ratio between the number of frequencies in F_R, F_G, F_B, and F_S is $2 : 2 : 2 : 1$.

Now we describe our algorithm CACO as follows:

2.2 Analysis

The high level idea to prove the performance of our algorithm CACO is to show that the ratio between the total number of accepted requests by CACO and the total number of satisfied requests by the optimal offline algorithm is at least 3/7. To prove this, we analyze the number of satisfied requests in each cell and its neighboring cells, then compare the number with the optimum value.

Algorithm 1. CACO: When a request arrives at a cell C with color $c \in \{R, G, B\}$

```
1: if F_c is not totally used up then
2:     assign the minimal available frequency from F_c to satisfy this request.
3: else if F_S is not totally used up in cell C and its neighboring cells then
4:     assign the minimal available frequency from F_S to satisfy this request.
5: else
6:     reject this request.
7: end if
```

Let R_i be the number of the requests arrived at cell C_i. Let O_i be the number of requests accepted by the optimal offline algorithm in cell C_i. $\sum O_i$ is the total number of accepted request by the optimal offline algorithm. Let A_i be the number of requests accepted by our online algorithm CACO in cell C_i. $\sum A_i$ is the total number of accepted request by CACO. Let $G_x(C_i)$ be the the number of requests accepted by CACO in cell C_i by assigning frequencies from F_x. It can be seen that $A_i = G_R(C_i) + G_G(C_i) + G_B(C_i) + G_S(C_i)$. If C_i is colored with $x \in \{R, G, B\}$, then $A_i = G_x(C_i) + G_S(C_i)$.

Fact 1. *For each cell C_i, $O_i \leq R_i$, $A_i \leq R_i$, and $A_i \geq 2\omega/7$ when $R_i \geq 2\omega/7$.*

According to the number of satisfied requests by the optimal offline algorithm, we classify the cells into two types: cell c_i is $safe$ if $O_i \leq 2\omega/3$, and *dangerous* otherwise.

Lemma 2. *If C_i is safe, then $A_i \geq 3O_i/7$*

Proof. This lemma can be proved by analyzing the following two cases.

- If $R_i \leq 2\omega/7$, $A_i = R_i \geq O_i$, then $A_i \geq 3O_i/7$.
- If $R_i > 2\omega/7$, CACO will accept at least $2\omega/7$ requests by assigning frequencies from F_x, thus, $A_i \geq 2\omega/7$. Since C_i is safe, $O_i \leq 2\omega/3$, thus, $A_i \geq 3O_i/7$.

□

Fact 3. *A safe cell has at most 3 dangerous neighboring cells. All neighboring cells around a dangerous cell are safe.*

Proof. This fact can be proved by contradiction. If a safe cell C has more than 3 dangerous neighboring cells, since C has 6 neighboring cells, there must exist two dangerous cells which are neighbors. From the definition of dangerous cell, the total number of accepted request in these two dangerous neighboring cells is strictly more than ω, contradiction!

Similarly, if a dangerous cell C' is a neighboring cell of another dangerous cell C, the total number of accepted request in C and C' is strictly more than ω. Contradiction! □

According to the algorithm, when a request cannot be satisfied in a cell C with color c, all frequencies in F_c must be used in C, and all frequencies in F_S must be used in C and its six neighbors. Thus, we have the following fact:

Fact 4. *If cell C cannot satisfy any request according to the algorithm CACO, then $G_S(C) + \sum G_S(C_k) \geq \omega/7$, where C_k is the neighboring cell of C.*

To compare the number of satisfied requests in each cell with the optimal offline solution, we define B_i as follows, where C_k represents the neighboring cell of C_i.

$$B_i = \begin{cases} 3O_i/7 & \text{if } C_i \text{ is safe} \\ A_i + \sum(A_k - 3O_k/7)/3 & \text{if } C_i \text{ is dangerous.} \end{cases}$$

Lemma 5. $\sum B_i \leq \sum A_i$.

Proof. According to Lemma 2, if C_i is safe, we have $A_i \geq 3O_i/7$. From Fact 3, we know there are at most three dangerous neighbors around C_i, thus, after counting $B_i = 3O_i/7$ frequencies in C_i, the remaining $A_i - 3O_i/7$ frequencies can compensate the frequencies in the dangerous neighboring cells, and each dangerous cell receives $(A_i - 3O_i/7)/3$ frequencies. From the definition of B_i, we can see that $\sum B_i \leq \sum A_i$. □

Theorem 1. *The asymptotic competitive ratio of algorithm CACO is at most 7/3.*

Proof. From the definition of O_i and B_i, we can say $O_i/B_i \leq 7/3$ for any cell leads to the correctness of this theorem. That is because $\sum O_i / \sum A_i \leq \sum O_i / \sum B_i \leq \max O_i/B_i$

If the cell is safe, i.e., $O_i \leq 2\omega/3$, we have $O_i/B_i = 7/3$.

If the cell C_i is dangerous, i.e., $O_i > 2\omega/3$, since $R_i \geq O_i > 2\omega/3 > 3\omega/7$, that means the number of requests R_i in this cell is larger than A_i. Thus, some requests are rejected in cell C_i, moreover, this cell cannot accept any further requests.

- If the number of accepted requests in any neighbor of C_i is no more than $2\omega/7$, we can say that all the shared frequencies in F_S are assigned to requests in cell C_i. Thus, $A_i = 3\omega/7$. We have

$$O_i/B_i = O_i/(A_i + (\sum(A_k - 3O_k/7))/3) \leq O_i/A_i \leq \omega/A_i = 7/3.$$

- Otherwise, suppose there are m neighbors of C_i in which the number of accepted requests are more than $2\omega/7$. Let $\hat{O}_i$ denote the average number of optimum value of accepted requests for these m neighboring cells around C_i.

$$
\begin{aligned}
B_i &= 2\omega/7 + G_S(C_i) + (\sum_{\text{for all safe neighbors}} (A_k - 3O_k/7))/3 \\
&\geq 2\omega/7 + G_S(C_i) + (m \times 2\omega/7 + \sum_{\substack{\text{for the neighbors} \\ \text{with } A_k > 2\omega/7}} G_S(C_k) - m \times 3\hat{O}_i/7)/3 \\
&\geq 2\omega/7 + (m \times 2\omega/7 + \sum_{\substack{\text{for the neighbors} \\ \text{with } A_k > 2\omega/7}} G_S(C_k) + G_S(C_i) - m \times 3\hat{O}_i/7)/3 \\
&= 2\omega/7 + (m \times 2\omega/7 + \omega/7 - m \times 3\hat{O}_i/7)/3 \\
&\geq 2\omega/7 + (2\omega/7 + \omega/7 - 3\hat{O}_i/7)/3 \\
&\quad \text{(that is because for any neighbor with } A_k > 2w/7, \\
&\quad O_k \leq (\omega - O_i) \leq \omega/3, \text{thus}, \hat{O}_i \leq \omega/3 \text{ and } 2\omega/7 - 3\hat{O}_i/7 \geq 0.) \\
&\geq 2\omega/7 + (3\omega/7 - 3(\omega - O_i)/7)/3 \\
&\quad \text{(since } O_k \leq \omega - O_i \text{ leads to } \hat{O}_i \leq \omega - O_i) \\
&= 2\omega/7 + O_i/7
\end{aligned}
$$

Thus, $O_i/B_i \leq O_i/(2w/7 + O_i/7) \leq 7/3$. □

In this kind of algorithms, the frequencies are partitioned into F_R, F_G, F_B and F_S, when a request arrives at a cell with color c, first choose the frequency from the set F_c, then from F_S if no interference appear. The performances are different w.r.t. the ratio between $|F_R|$ ($|F_G|$, $|F_B|$) and $|F_S|$. Note that from symmetry, the size of F_R, F_G and F_B should be same. Now we show that CACO is best possible among such kind of algorithms. Suppose the ratio between $|F_R|$ and $|F_S|$ is $x : y$. Consider the configuration shown in Figure 2. In the first step, ω requests arrive at the center cell C with color c, the algorithm will use up all frequencies in F_c and F_S, in this case, the ratio of accepted requests by the optimal offline algorithm and the online algorithm is $(3x + y)/(x + y)$ since the optimal algorithm will accept all these requests. In the second step, ω requests arrive at C_1, C_2 and C_3 with the same color c'. The online algorithm can only accept $x\omega/(3x + y)$ requests in each C_i $(1 \leq i \leq 3)$ since the frequencies in F_S are all used in C. In this case, the ratio between the optimal offline algorithm and the online algorithm is $3(3x + y)/(4x + y)$ since the optimal algorithm will accept all ω requests in C_i $(1 \leq i \leq 3)$ and reject all requests in C. Balancing these two ratios, we have $x : y = 2 : 1$, and the ratio is at least $7/3$.

3 Call Control in Triangle-Free Cellular Networks

The call control problem in cellular network is hard. But for some various graph classes, this problem may have a better performance. For example, in linear network, an optimal online algorithm with competitive ratio $3/2$ can be achieved [10]. An interesting induced network, *triangle-free cellular network*, has been studied for many problems including frequency assignment problem[4,9].

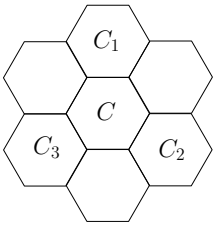
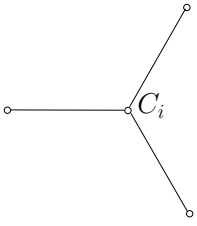
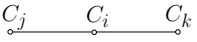


Fig. 2. Algorithm CACO is best possible among this kind of algorithms



(a) Structure A: neighbors with the same base color



(b) Structure B: neighbors with different base colors

Fig. 3. Structure of neighboring cells

For a given cell C_i, from the definition of triangle-free, only two possible configurations may exist for the structure of neighboring cells, which are shown in Fig. 3. It is easy to see that if C_i has 3 neighbors, the neighboring vertices are of the same color. On the other hand, if the neighbors are of different colors, C_i has at most 2 neighbors. There exists a simple structure in triangle-free cellular network, i.e., a cell has only one neighbor, we can regard this structure as the case in Fig. 3(b).

For the three base colors R, G and B, we define a cyclic order among them as $R \to G$, $G \to B$ and $B \to R$. Partition the frequency set $\{1, ..., \omega\}$ into three disjoint sets:

$$F_R = \{1, ..., \omega/3\},\ F_G = \{\omega/3 + 1, ..., 2\omega/3\},\ F_B = \{2\omega/3 + 1, ..., \omega\}$$

To be precisely, assigning frequencies from a set must in order of *bottom-to-top* (assigning frequencies from lower number to higher number) or *top-to-bottom* (assigning frequencies from higher number to lower number). Now we describe our algorithm for call control problem in triangle-free cellular networks.

Algorithm CACO2: Handling request in a cell C with color $X \in \{R, G, B\}$

1. If cell C has no neighbors, just assign frequency from 1 to ω.

2. If cell C has neighboring structure A (Fig. 3(a)), let Y be the base color of C's neighbors and Z be the other third color. Assign frequency in cell C as follows if no interference appear:
   (a) Assign frequencies from F_X in bottom-to-top order.
   (b) If all frequencies in F_X are used up, assign frequencies from F_Z in bottom-to-top order if $X \to Y$; and in top-to-bottom order otherwise. Such assignment can make sure if C uses the frequency from F_Z after using up all frequencies from F_X, and its neighboring cell C' also uses the frequency from F_Z after using up the frequencies from F_Y, C and C' must assign frequency from F_Z in different order no matter the neighbor configuration of C' is. (This can be verified by checking this case (case 2) and the next case (case 3) of CACO2.)
3. If cell C has neighboring configuration B (Fig. 3(b)), let Y and Z be the base colors of its two neighbors, respectively. Without loss of generality, assume $X \to Y$. Assign frequency in cell C as follows if no interference appear:
   (a) Assign frequencies from F_X in bottom-to-top order.
   (b) If all frequencies in F_X are used up, assign frequencies from F_Y in top-to-bottom order.

Theorem 2. *The competitive ratio of CACO2 is at most 9/4.*

Proof. Assume at some time, let O_i and A_i denote the number of accepted requests in cell C_i by the optimal offline algorithm and online algorithm CACO2, respectively. The theorem holds if $\sum O_i / \sum A_i \leq 9/4$. Similar to the analysis for CACO, we define B_i as the amortized number of accepted requests in cell C_i. Thus, our target is to prove that $O_i/B_i \leq 9/4$ and $\sum B_i \leq \sum A_i$. W.l.o.g., let X, Y and Z denote the three colors in the network.

Intuitively, we may set $B_i = 4O_i/9$ if $A_i \geq 4O_i/9$ in cell C_i, and the remaining uncounted frequencies can be used to compensate the number of accepted frequencies in its neighboring cells. Next, we describe how to partition the remaining uncounted frequencies according to cell C_i's neighboring configuration. Let H_{ij} to be the number of frequencies used in C_i but will compensate the number of frequencies in C_j.

1. The neighboring configuration of C_i is A (Fig. 3(a)), the uncounted number of frequencies is $A_i - 4O_i/9$, evenly distribute this number to the three neighboring cells, i.e., each neighboring cell C_j of C_i receives $H_{ij} = (A_i - 4O_i/9)/3$.
2. The neighboring configuration of C_i is B (Fig. 3(b)), denote the color of C_i to be X, and the colors of its neighboring cells to be Y (cell C_j) and Z (cell C_k) respectively. W.l.o.g., assume $X \to Y$, $Y \to Z$ and $Z \to X$.
   – If $A_i > \omega/3$,
     In this case, the requests in cell C_i will use some frequencies from F_Y. If $A_j < 4O_j/9$, there exist rejected request in C_j, thus, $A_i + A_j = 2\omega/3$. The remaining uncounted number of frequencies in C_i can be partitioned into $(4O_j/9 - A_j)$ and $\omega/9$, the former part $(4O_j/9 - A_j)$ compensates the number in C_j (i.e., $H_{ij} = 4O_j/9 - A_j$ if $A_j < 4O_j/9$) and the later

part $\omega/9$ compensates the number in C_k (i.e., $H_{ik} = \omega/9$ if $A_k < 4O_k/9$). This compensation is justified since $4O_i/9+(4O_j/9-A_j)+\omega/9 = 4(O_i+O_j)/9 - A_j + \omega/9 \leq 5\omega/9 - A_j < A_i$.

– If $A_i \leq \omega/3$,
In this case, all frequencies used in C_i are from F_X, and some frequencies used in C_k may from F_X too. If $A_k < 4O_k/9$, all remaining uncounted number $A_i - 4O_i/9$ of frequencies in C_i will compensate the number in C_k, i.e., $H_{ik} = A_i - 4O_i/9$ and no extra number of frequencies compensates the number of frequencies in C_j, i.e., $H_{ij} = 0$.

We define B_i as follows, where H_{ji} is the compensation from neighboring C_j.

$$B_i = \begin{cases} 4O_i/9 & \text{if } A_i \geq 4O_i/9 \\ A_i + \sum H_{ji} & \text{if } A_i < 4O_i/9, \end{cases}$$

From previous description, we can say that $4O_i/9 + \sum_j H_{ij} \leq A_i$ if $A_i \geq 4O_i/9$, thus,

$$\begin{aligned} \sum B_i &= \sum_{A_i \geq 4O_i/9} 4O_i/9 + \sum_{A_i < 4O_i/9} (A_i + \sum_{C_i \text{ and } C_j \text{ are neighbors}} H_{ji}) \\ &= \sum_{A_i \geq 4O_i/9} (4O_i/9 + \sum_{C_i \text{ and } C_j \text{ are neighbors}} H_{ij}) + \sum_{A_i < 4O_i/9} A_i \\ &\leq \sum_{A_i \geq 4O_i/9} A_i + \sum_{A_i < 4O_i/9} A_i \\ &\leq \sum_i A_i \end{aligned}$$

Now we analyze the relationship between B_i and O_i. Assuming the color of C_i is X.

1. If $A_i \geq 4O_i/9$, $B_i = 4O_i/9$.
2. If $A_i < 4O_i/9$,
   (a) If $A_i < \omega/3$
   Since $A_i < 4O_i/9$, there must exist some rejected requests in C_i. Some frequencies in F_X are used in one of C_i's neighbor C_j. According to the algorithm, the neighboring structure of C_j is B (Fig. 3(b)), and $A_i + A_j = 2\omega/3$.
   In this case, $H_{ji} = 4O_i/9 - A_i$, thus,

$$B_i = A_i + \sum_{C_k \text{ and } C_i \text{ are neighbors}} H_{ki} \geq A_i + H_{ji} = 4O_i/9.$$

   (b) If $A_i \geq \omega/3$ and C_i has two neighbors C_j with color Y and C_k with color Z as shown in Fig. 3(b). W.l.o.g., assume that $X \to Y$, $Y \to Z$ and $Z \to X$. According to the algorithm, after using up the frequencies in F_X, C_i will use some frequencies from F_Y until interference appear, thus, $A_i + A_j \geq 2\omega/3$.

i. If the neighboring configuration around C_j is A (Fig. 3(a)), we claim that $A_j \geq 4O_j/9$. That is because $O_j \leq \omega - O_i \leq \omega - 9A_i/4 \leq \omega - 9\omega/12 = \omega/4$, $A_i \leq 4O_i/9 \leq 4\omega/9$, and $A_i + A_j \geq 2\omega/3$.
In this case, $H_{ji} = (A_j - 4O_j/9)/3$, and

$$B_i \geq A_i + H_{ji} = A_i + (A_j - 4O_j/9)/3 \geq 4O_i/9.$$

ii. If the neighboring configuration around C_j is B (Fig. 3(b)),
– if $A_j \leq \omega/3$, we have $H_{ji} = A_j - 4O_j/9$. Thus,

$$B_i \geq A_i + H_{ji} = A_i + A_j - 4O_j/9 \geq 2\omega/3 - 4O_j/9 \geq 4O_i/9.$$

– If $A_j \geq \omega/3$, $H_{ji} = \omega/9$, thus,

$$B_i \geq A_i + H_{ji} = A_i + \omega/9 \geq \omega/3 + \omega/9 = 4\omega/9 \geq 4O_i/9.$$

(c) If $A_i \geq \omega/3$ and the neighbors of C_i are of the same color (Fig. 3(a)), assume the color of its neighboring cell is Y. According to the algorithm, after using up the frequencies from F_X, C_i will use some frequencies from F_Z to satisfy some requests. Since C_i rejects some requests, we have $A_i + A_j = \omega$ for some neighboring cell C_j of C_i. This is because C_i and C_j assign frequencies from F_Z in different order, and C_j will use the frequency from F_Z after using up the frequency from F_Y.
In this case, $H_{ji} = (A_j - 4O_j/9)/3$ if the neighboring configuration of C_j is A (Fig. 3(a)), or $H_{ji} = \omega/9$ if the neighboring configuration of C_j is B (Fig. 3(b)). In the former case,

$$B_i \geq A_i + H_{ji} = A_i + (A_j - 4O_j/9)/3 > 4O_i/9;$$

in the later case,

$$B_i \geq A_i + H_{ji} = A_i + \omega/9 \geq 4\omega/9 \geq 4O_i/9.$$

Combine all above cases, we have $O_i/B_i \leq 9/4$ in each cell C_i. Since $\sum B_i \leq \sum A_i$, we have $\sum O_i / \sum A_i \leq 9/4$. □

Next, we show that the lower bound of competitive ratio for call control problem in triangle-free cellular networks is at least 5/3.

Theorem 3. *The competitive ratio for call control problem in triangle-free cellular network is at least 5/3.*

Proof. We prove the lower bound by using an adversary who sends request according to the assignment of the online algorithm.

Consider the configuration shown in Figure 4.

In the first step, the adversary sends ω requests in the center cell C. Suppose the online algorithm accepts x requests. If $x \leq 3\omega/5$, the adversary stop sending request. In this case, the optimal offline algorithm can accept all these ω requests, thus, the ratio is at least 5/3.

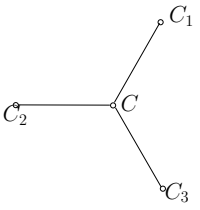


Fig. 4. lower bound of competitive ratio is at least 5/3

If $x > 3\omega/5$, the adversary then sends ω requests in each cell of C_1, C_2 and C_3. To avoid interference, the online algorithm can accept at most $\omega - x$ requests in each cell, and the total number of accepted requests is $x + 3(\omega - x) = 3\omega - 2x$. In this case, the optimal offline algorithm will accept 3ω requests, i.e., reject all requests in the center cell C. Thus, the ratio in this case is $3\omega/(3\omega - 2x)$. Since $x > 3\omega/5$, this value is at least 5/3.

Combine the above two cases, we can say that the competitive ratio is at least 5/3. □

References

1. Caragiannis, I., Kaklamanis, C., Papaioannou, E.: Efficient on-line frequency allocation and call control in cellular networks. Theory Comput. Syst. 35(5), 521–543 (2000); A preliminary version of the paper is in SPAA 2000 (2000)
2. Caragiannis, I., Kaklamanis, C., Papaioannou, E.: Competitive Algorithms and Lower Bounds for On-Line Randomized Call Control in Cellular Networks. Networks 52(4), 235–251 (2008); Preliminary versions are in WAOA 2003 and EUROPAR 2005
3. Chan, W.-T., Chin, F.Y.L., Ye, D., Zhang, Y.: Online Frequency Allocation in Cellular Networks. In: Proc. of the 19th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2007), pp. 241–249 (2007)
4. Havet, F.: Channel assignment and multicoloring of the induced subgraphs of the triangular lattice. Discrete Math. 233, 219–231 (2001)
5. MacDonald, V.H.: Advanced mobile phone service: The cellular concept. Bell Systems Technical Journal 58(1), 15–41 (1979)
6. McDiarmid, C., Reed, B.: Channel assignment and weighted coloring. Networks 36(2), 114–117 (2000)
7. Narayanan, L., Shende, S.: Static frequency assignment in cellular networks. Algorithmica 29(3), 396–409 (2001)
8. Pantziou, G.E., Pentaris, G.P., Spirakis, P.G.: Competitive Call Control in Mobile Networks. Theory of Computing Systems 35(6), 625–639 (2002)
9. Šparl, P., Žerovnik, J.: 2-local 5/4-competitive algorithm for multicoloring triangle-free hexagonal graphs. Inf. Process. Lett. 90, 239–246 (2004)
10. Ye, D., Han, X., Zhang, G.: Deterministic On-line Call Control in Cellular Networks (manuscript)
11. Zhang, Y., Chin, F.Y.L., Zhu, H.: A 1-Local Asymptotic 13/9-Competitive Algorithm for Multicoloring Hexagonal Graphs. Algorithmica 54, 557–567 (2009)

Online Algorithms for the Newsvendor Problem with and without Censored Demands

Peter Sempolinski and Amitabh Chaudhary

University of Notre Dame, Notre Dame IN 46556, USA
{psempoli,achaudha}@nd.edu

Abstract. The newsvendor problem describes the dilemma of a newspaper salesman—how many papers should he purchase each day to resell, when he doesn't know the demand? We develop approaches for this well known problem in operations research, both for when the actual demand is known at the end of each day, and for when just the amount sold is known, i.e., the demand is *censored*. We present three results: (1) the first known algorithm with a bound on its worst-case performance for the censored demand newsvendor problem, (2) an algorithm with improved worst-case performance bounds for the regular newsvendor problem compared to previously known algorithms, and (3) more precise bounds on the performance of the two algorithms when they are seeded with an approximate "guess" on the optimal solution. In addition (4) we test the algorithms in a variety of simulated and real world conditions, and compare the results to those by previously known approaches. Our tests indicate that our algorithms perform comparably and often better than known approaches.

1 Introduction

The newsvendor problem is about deciding the number of items of a product a vendor should order to meet an unpredictable demand, when the product has a short life-cycle like newspapers, perishables, or fashion apparel. Formally, in this problem we receive, *online*, a sequence $\sigma = d_1, \ldots, d_T$ of demands, where each d_i is from a given interval $[m, M]$ of valid demands. Before we can see demand d_i, we have to place an order amount x_i. We only actually sell the lesser of d_i and x_i; i.e., our profit is $r \min\{d_i, x_i\} - cx_i$ where r is the per unit revenue and c is the per unit cost. Any unsold items go waste, and cannot be used in the next step, and similarly any unmet demand is lost and not seen again. The objective is to place orders that maximize the total profit over the demand sequence.

There is a more realistic *censored demands* version of the problem in which the true demand of a product is not revealed to the vendor, instead just the actual sales is. Here, at each step i, after we order amount x_i, we learn the amount sold $s_i = \min\{d_i, x_i\}$, and earn the profit $rs_i - cx_i$. These two newsvendor problems play a fundamental role in real-world supply chain planning of several kinds of products, such as consumer electronics, seasonal products, perishables, and certain kinds of vaccines. They have been extensively studied within operations

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 234–249, 2010.

research, although almost always under the assumption that the demands are *independently and identically distributed* according to some underlying distribution, which is known to a varying degree (see, e.g., [4, 8, 13, 17, 23, 31, 32, 34, 35, 37, 38, 39, 40, 41, 44]).

In this paper we present three results: (1) the first known algorithm with a bound on its worst-case performance for the censored demand newsvendor problem, (2) an algorithm with improved worst-case performance bounds for the regular newsvendor problem compared to previously known algorithms, and (3) more precise bounds on the performance of the two algorithms when they are seeded with an approximate "guess" on the optimal solution. Further (4) we present results of extensive tests of our algorithms on simulated and real data.

Offline Adversaries for the Newsvendor. A common approach taken to resolve the demand uncertainty issue is using a stochastic model for the demands; assuming, for example that for each period the demand is drawn independently from some known distribution. In using such an approach, the goal is then to choose an order amount which maximizes expected profit (see, e.g., [12]). However, such approaches are commonly inadequate, as the quality of the final result depends heavily on the quality of the assumptions made about the distribution. Given the strong uncertainty inherent in many newsvendor items, the quality of such approaches is usually low. (See [44] for a lengthier discussion on the shortcomings of this approach.) We observe this as well in our experiments on real-world data (see Section 4).

Alternate approaches to the newsvendor problem are more "adversarial" in nature. In these models, very little is assumed about the nature of the demands, and worst-case analysis is used. Typically, only a lower bound m and upper bound M on the range of possible demand values are assumed. Here, one possible approach is to minimize the *maximum regret*:

$$\max_{\text{demand sequences}} (\mathsf{OPT} - \mathsf{ALG}),$$

where OPT denotes the profit of the *offline optimal* algorithm which knows the demand sequence in advance (see, e.g., [5]), and ALG is the profit of the strategy used (see [38, 44, 46]). This is related to the standard approach for evaluating and designing online algorithms in computer science, i.e., to minimize the *competitive ratio*, which is roughly the ratio $\mathsf{OPT}/\mathsf{ALG}$ in the worst case (see [5]).

The above approaches are not useful, however, since using Yao's technique, one can prove a lower bound of $Tc(M-m)(r-c)/r$ on the maximum regret, and similarly a lower bound of $Mc/(mr) + (r-c)/r$ on the competitive ratio for any randomized online algorithm [31, 32]. Such "high" lower bounds are not surprising since here an algorithm has to make a decision corresponding to an input *before* the input is received. This is unlike online problems in, say, caching or scheduling, where the decision is made after the input is received but before the next input is received.

For this reason, we turn away from evaluating the performance of algorithms in terms of the *dynamic* offline optimal, and consider a more realistic target:

the *static* offline optimal, which we denote here by STOPT. STOPT is a weaker version of OPT which makes an optimal decision based on perfect knowledge of the demands, but is required to choose one single order quantity to use for all periods. This approach is standard in computational learning theory and online learning [7, 19]. Bertsimas and Thiele [4] show that on a sequence of T demands, STOPT chooses to order, every day, an amount equal to the demand of rank $\lceil T - Tc/r \rceil$ in the sorted non-decreasing sequence of the demands. In particular, we evaluate our algorithms using *maximum regret from static optimal*:

$$\max_{\text{demand sequences}} (\mathsf{STOPT} - \mathsf{ALG}),$$

Comparing the performance of algorithms with the performance of STOPT is both theoretically and practically significant, because *any bounds for an algorithm with respect to* STOPT *also hold with respect to an algorithm which makes decisions based on stationary stochastic assumptions*—assumptions made by most of the algorithms in the inventory theory literature[8, 12, 13, 17, 23, 34, 35, 38, 39, 40, 41, 44].

1.1 Our Contributions

In this paper we present three algorithms, bounds on their worst-case performance in terms of regret from STOPT, and their empirical evaluation on simulated as well as real data. Our contributions are—

- A simple deterministic algorithm for the regular newsvendor problem with a bound on regret from STOPT. This is in contrast to the bound on the algorithm's regret from OPT in terms of STOPT's regret from OPT given in [32, 33]—the previously known worst-case bound for the problem. (Section 3.2.)
- An simple deterministic algorithm for the censored demand newsvendor problem with the first known worst-case bound—on regret from STOPT. (Section 3.4.)
- Versions of the two algorithms above that accept a guess on the STOPT solution, and more precise bounds on their regret from STOPT based on the quality of the guess. (Section 3.3.)
- Extensive empirical comparison of the performance of our algorithms with known algorithms from operations research and inventory management on simulated and real data from a supermarket. (Section 4.)

We begin with an brief outline of the extensive past work on these problems, and other related work in machine learning in Section 2.

2 Related Work

The Newsvendor Problem. The newsvendor problem is easy to solve when we know the true demand distribution, in which case we can apply the well-known

"critical fractile" solution: in each period order x where $\phi(x) = (r-c)/r$ and $\phi(\cdot)$ is the cumulative probability function for the distribution [35]. This maximizes the expected profit.

In most real-life scenarios, however, the true demand distribution is not available and so several studies assume that we have only partial information on the underlying distribution. In this, the *Bayesian* approach is most popular. It assumes knowledge of the family of distributions to which the true distribution belongs, but not of the specific parameters. It begins with an prior belief about the parameter values, and continually updates its belief based on observations of demand over time, by computing posterior distributions. See, e.g., work by Scarf [39, 40], Karlin [21], Iglehart [18], Murray and Silver [29], and Azoury [2]. In several applications, however, updating the prior distribution is found to be expensive [30].

A contrasting approach is the *non-parametric* or *robust* approach, as taken by the well-known Scarf's Rule, which maximizes the expected profit for the worst-case demand distribution, when only the mean and variance of the distribution are known [13, 41]. This *maximin* approach is, however, risk averse [34]. A slightly less conservative approach is that of *minimax expected regret*, which aims to minimize the maximum expected loss from not being able to make optimal decisions because of limited demand information [3, 8, 24, 34, 38, 44]. (We use a similar criteria, but in the context of adversaries, minimizing the maximum regret over all sequences, rather than the maximum expected regret).

A different approach is that of *sample average approximation* taken by Levi et al. [23]. Their policy is based only on the observed samples of demands and, when the number of samples is larger than a specific lower bound, is guaranteed to have a profit arbitrarily close to that of the policy that knows the true distribution.

All of the above policies assume that the demands follow a single (possible unknown) underlying distribution, in particular, they are *independent and identically distributed*. This itself need not be true in real-world situations.

There has been some work which does not assume an underlying distribution. With respect to OPT, O'Neil [31] gives simple deterministic "balancing" algorithms that match the lower bounds on the maximum regret, $Tc(M-m)(r-c)/r$, and competitive ratio, $Mc/(mr) + (r-c)/r$. More importantly, O'Neil et al. [31, 32, 33] give an algorithm WMN based on the weighted majority approach [25], that has a bound on its regret from the dynamic optimal OPT in terms of the regret of STOPT from OPT. They also give an algorithm WMNS which has a similar bound except that the static optimal in its case is "shifting", i.e., allowed to change the order quantity a fixed number of times during the sequence.

Recently, Zhang and Xu [47] have analyzed the newsvendor problem using the risk-reward version of competitive analysis introduced by [1]. Bertsimas and Thiele [4] give solutions for several variants of the newsvendor problem which optimize the order quantity purely based on historical data, including the amount ordered each day by STOPT. They also take into account risk preferences by "trimming" historical data which would lead to overly optimistic predictions.

The Censored Demands Newsvendor Problem. There are several papers that follow the Bayesian approach for the censored demands newsvendor problem, beginning with Conrad [10], and then by Harpaz et al. [15], Ding et al. [11], and Lu et al. [27, 28]. Also related are papers by Chen and Plambeck [9], Lariviere and Porteus [22], and Liyanage and Shanthikumar [26]. These latter papers, however, consider the variant when the inventory is non-perishable and carries over to the next period.

Following the non-parametric approach, Burnetas and Smith [6] give a stochastic approximation algorithm that approximates the newsvendor quantile of the demand distribution. Huh and Rusmevichientong [17] and Huh et al. [16] apply stochastic online convex optimization techniques (a stochastic variant of the classical gradient descent method) to this problem. They give policies whose per period performance converges, as T increases, to that of the policy that knows the underlying distribution. Godfrey and Powell [14] and Powell et al. [36] present a non-parametric method CAVE to successively approximate the convex objective function with piecewise linear functions. (We compare our algorithms with CAVE in Section 4.)

Online Decision Making. Our basic approach is somewhat related to the Algorithm "Follow the Perturbed Leader," a general algorithm for online decision making, also applicable to the learning from experts problem. It's creators, Kalai and Vempala [20], apply the algorithm to such problems as online shortest paths [43] and the tree update problem [42]. Their algorithm, however, relies on randomization to bound the performance. Our algorithms are deterministic and focus on the specific structure of the newsvendor problems.

3 Algorithms and Analysis

In this section we present a basic incremental algorithm INC, which has a bounded regret from STOPT. We then make this bound more precise by modifying it into algorithm INCβ, which adds imaginary initial demands to input sequence. The precision of the bound is improved by controlling the differences ("gaps") between the imaginary demands added, as well as the STOPT solution for that initial demand sequence. Lastly, we show how the above algorithms can be modified to allow for bounds when the demands are censored.

Denote the demand sequence by $\sigma = d_1, \ldots, d_T$, the revenue per item by r and the cost per item by c. Denote the subsequence of demands till step i, for $i \leq T$, by $\sigma(i) = d_1, \ldots, d_i$. Further, denote the *$j$-order statistic* of the demands by $d^{(j)}$. (The j-order statistic is the demand at rank (position) j in the sequence of demands sorted in non-decreasing order [45].) Recall that given σ, STOPT chooses to order the amount $d^{(\lceil T - Tc/r \rceil)}$ [4]. Denote the profit of algorithm ALG on σ by ALG(σ). Lastly, if β and σ are two demand sequences, denote the concatenated sequence by (β, σ).

3.1 Basic Incremental Algorithm

The basic incremental algorithm INC is best described by first describing the following related, albeit hypothetical, algorithm INC$\mathcal{L}$ which is online but has the benefit of a *lookahead* of one: before deciding the order amount at step i, INC$\mathcal{L}$ knows all demands in $\sigma(i)$.

Algorithm INC$\mathcal{L}$. At step i order the amount that STOPT orders on sequence $\sigma(i)$, i.e., the $\lceil i - ic/r \rceil$-order statistic of $\sigma(i)$.

Lemma 1. $\mathsf{INC}\mathcal{L}(\sigma) \geq \mathsf{STOPT}(\sigma)$.

Proof. The proof is by induction on the number of steps i considered in the sequence. For the base case of $i = 1$, it is clear that both INC$\mathcal{L}$ and STOPT order the single demand value. For the inductive step assume $\mathsf{INC}\mathcal{L}(\sigma(i)) \geq \mathsf{STOPT}(\sigma(i))$. Let the amount ordered by STOPT on $\sigma(i+1)$, and thus the amount ordered by INC$\mathcal{L}$ on step $(i+1)$ on σ, be x. Thus $\mathsf{INC}\mathcal{L}(\sigma(i+1)) = \mathsf{INC}\mathcal{L}(\sigma(i)) + r\min\{d_{i+1}, x\} - cx$, which, by induction, is at least $\mathsf{STOPT}(\sigma(i)) + r\min\{d_{i+1}, x\} - cx$.

We claim that $\mathsf{STOPT}(\sigma(i))$ is at least as large as the profit of ordering x for each step of $\sigma(i)$, i.e., the profit STOPT, when operating on $\sigma(i+1)$, earns on the first i steps. This follows directly from the optimality of $\mathsf{STOPT}(\sigma(i))$. Combining this the result above we get $\mathsf{INC}\mathcal{L}(\sigma(i+1)) \geq \mathsf{STOPT}(\sigma(i+1))$.

Algorithm INC. At step 1 order m. At step $i \geq 2$ order the $\lceil i - ic/r \rceil$-order statistic of $\sigma(i-1)$, if it exists, and order the $(i-1)$-order statistic of $\sigma(i-1)$ otherwise.

Denote by Δ_i the maximum difference between two demands of consecutive ranks in $\sigma(i)$ (padded with an initial demand of m), i.e., $\max_{j \geq 0}(d_{(i)}^{(j+1)} - d_{(i)}^{(j)})$, where $d_{(i)}$ refers to demands in $\sigma(i)$ and $d_{(i)}^{(0)} = m$. Further let Δ denote $\max_i \Delta_i$. We now bound the performance of INC.

Lemma 2. $\mathsf{INC}(\sigma) \geq \mathsf{STOPT}(\sigma) - T\Delta\max\{r-c, c\}$.

Proof. We show that $\mathsf{INC}(\sigma) \geq \mathsf{INC}\mathcal{L}(\sigma) - T\Delta\max\{r-c, c\}$. The rest follows from Lemma 1.

Observe that if at some step INC$\mathcal{L}$ orders x and INC orders $x \pm \delta$, then as a result INC's profit can be lower than INC$\mathcal{L}$'s by at most $\max\{x(r-c) - (x-\delta)(r-c), x(r-c) - (xr - (x+\delta)c)\} = \delta\max\{r-c, c\}$.

We now show that at each step i, the difference between the order of INC$\mathcal{L}$ and the order of INC is Δ_i, which will complete the proof. This is clear for step $i = 1$, in which INC orders m and INC$\mathcal{L}$ orders d_1. For step $i \geq 2$, first assume that the $\lceil i - ic/r \rceil$-order statistic exists in $\sigma(i-1)$. Observe that both INC$\mathcal{L}$ and INC consider this order statistic, but INC$\mathcal{L}$ considers it in the sequence $\sigma(i)$ while INC considers it in the the shorter sequence $\sigma(i-1)$. Depending on the rank of d_i in $\sigma(i)$, these amounts may be the same, or differ by at most Δ_i. When the $\lceil i - ic/r \rceil$-order statistic does not exist in $\sigma(i-1)$, it has to be the demand at

rank i. In that case INC orders the $(i-1)$-order statistic in $\sigma(i-1)$ and INC$\mathcal{L}$ orders the i-order statistic in $\sigma(i)$, and again the two order amounts can differ by at most Δ_i.

3.2 Imaginary Initial Demands

Algorithm INC's bound depends on Δ, which for some input sequences may be as large as $(M-m)$. To control Δ, we add an imaginary sequence of initial demands $\beta = b_1, \ldots, b_t$ to the *beginning* of the input; so on input sequence σ, the algorithm imagines it receives the input (β, σ). We show how this leads to an algorithm INCβ which has tighter bounds.

Let β be such that its demands are some δ apart and "cover" the range $[m, M]$, i.e., $\beta = m, m+\delta, m+2\delta, \ldots, M$. It is used in the following algorithm.

Algorithm INCβ. Given a sequence of beginning demands β, run INC on the sequence (β, σ). The profit INCβ earns, however, is the profit on the actual demands in σ.

Since β occurs at the beginning it ensures that, irrespective of σ, $\Delta_i \leq \delta$ for every step i of input (β, σ). Thus $\Delta \leq \delta$ for the entire input. Introducing β does, however, affect the algorithm's performance in other ways, which we bound below. We begin with some notation.

Let σ' be a subsequence of σ. When an algorithm ALG runs on input σ, denote the profit it earns on just σ' by ALG$(\sigma' \mid \sigma)$. Thus INC$\beta(\sigma) =$ INC$(\sigma \mid (\beta, \sigma))$. Further, let x be the order amount chosen by STOPT when it runs on input σ. If STOPT uses x on an input sequence β, denote the profit it earns by STOPT$(\beta \mid \sigma)$.

Lemma 3. INC$\beta(\sigma) \geq$ STOPT$(\sigma) - ((T+t)\delta \max\{r-c, c\} +$ INC$(\beta) -$ STOPT$(\beta \mid \sigma))$, *in which* T *and* t *are the lengths of* σ *and* β, *respectively, and* δ *is the* Δ*-value for* (β, σ).

Proof. Clearly, INC$\beta(\sigma)$, which is INC$(\sigma \mid (\beta, \sigma))$, equals INC$(\beta, \sigma) -$ INC$(\beta \mid (\beta, \sigma))$. Since INC is online INC$(\beta \mid (\beta, \sigma)) =$ INC(β).

Using Lemma 2, we can lower bound INC(β, σ) by STOPT$(\beta, \sigma) - (T + t)\delta \max\{r - c, c\}$. Let x be the order amount chosen by STOPT on input σ. From its optimality, STOPT(β, σ) is at least the profit if x is chosen in all steps of input (β, σ), i.e.,

$$\text{STOPT}(\beta, \sigma) \geq \text{STOPT}(\beta \mid \sigma) + \text{STOPT}(\sigma).$$

Combining the results above completes the proof.

At each step in β, the difference between the order amount INC places and x can be at most $(M-m)$. This leads to the following theorem.

Theorem 1. INC$\beta(\sigma) \geq$ STOPT$(\sigma) - ((T+t)\delta \max\{r-c, c\} + t(M-m) \max\{r-c, c\})$, *in which* T *and* t *are the lengths of* σ *and* β, *respectively, and* δ *is the* Δ*-value for* (β, σ).

Observe that the last term is a constant in terms of T. Thus if $\delta = \Theta((M-m)/\sqrt{T})$ is chosen, we get INC$\beta(\sigma) \geq$ STOPT$(\sigma) - O(\sqrt{T}(M-m) \max\{r-c, c\})$.

3.3 Tighter Bounds for Reasonable Guesses

We can, however, get tighter bounds if β is chosen more carefully. We first show how to choose β such that $\mathsf{INC}(\beta) = \mathsf{STOPT}(\beta)$. Let X be target order amount we want STOPT to choose on input β. We choose the demands b_i in β such that INC also orders X at each step. Observe that if on some step INC chooses the order statistic k, then on the next it either chooses the same order statistic k or the next $(k+1)$. This is because for any i, $\lceil i - ic/r \rceil$ is at most 1 larger than $\lceil (i-1) - (i-1)c/r \rceil$, when $c/r < 1$. We sketch how to construct β below.

Modify INC to choose X at step 1 (instead of the earlier specified m). Choose $b_1 = X$. This ensures that INC chooses X in step 2. The remaining b_i values "move away" from X in steps of δ, until they "cover" the range $[m, M]$. At each step i we add a b_i which is the next higher or next lower based on the following cases:

- *Case 1:* If $\lceil i - ic/r \rceil = \lceil (i-1) - (i-1)c/r \rceil$, add the next higher value of b_i.
- *Case 2:* Else, if $\lceil i - ic/r \rceil = \lceil (i-1) - (i-1)c/r \rceil + 1$, we add the next lower values of b_i.

Both cases have to occur since $c/r < i$. If, however, say the lower range is covered before the higher, we simply repeat b_i equal to m until the higher is covered as well, and vice versa. This ensures that INC orders X at every step on β.

For the rest of the paper we assume β is constructed using the method above. Clearly, if X is x, the amount STOPT orders on input σ, then $\mathsf{INC}(\beta) - \mathsf{STOPT}(\beta \mid \sigma) = 0$, and the adverse effect of adding β is neutralized. But since x is not known in advance we guess at value X. We bound the the profit of $\mathsf{INC}\beta$ in terms of the difference $|X - x|$.

Theorem 2. $\mathsf{INC}\beta(\sigma) \geq \mathsf{STOPT}(\sigma) - ((T+t)\delta \max\{r-c, c\} + O(|X-x|^2 r/\delta))$, *in which T and t are the lengths of σ and β, respectively, and δ is the Δ-value for (β, σ).*

Proof. Denote $\mathsf{INC}(\beta) - \mathsf{STOPT}(\beta \mid \sigma)$ by R. We begin with bounding the change in R when $|X - x|$ is increased by 1. Assume $X \leq x$. Let the number of b_i values in the interval $[X, x]$ be q. It is easy to see that on decreasing X by 1, the increase in R is $r(\lfloor tc/r \rfloor + q) - ct = O(qr)$. A similar argument shows that when $X > x$ and X is increased by 1, the increase in R is $O(qr)$.

Since b_is are δ apart, $q = O(|X - x|/\delta)$. Furthermore, since $R = 0$ when $X = x$, it follows by "integrating" over the distance $|X - x|$ that R itself is $O(|X - x|^2 r/\delta)$.

In practice, we found that varying the value of X does indeed result in a parabolic curve for the profit on $\mathsf{INC}\beta$. Further, $\mathsf{INC}\beta$ performs well as long as X is a reasonable guess of x.

3.4 Censored Demands

In the real world demands are usually censored, i.e., at each step i only the amount of sales $s_i = \min\{d_i, x_i\}$ is revealed to the algorithm. We show how we can

modify INC and INCβ into algorithms for censored demands, $\mathcal{C}$INC and $\mathcal{C}$INCβ, respectively, which can guarantee a minimum profit in terms of STOPT(σ), the optimal profit on the *uncensored* demand sequence. Let the censored demand sequence for the algorithm in question be represented by cen(σ).

Algorithm $\mathcal{C}$INC. On input cen(σ) proceed as algorithm INC would on input cen(σ), except if at any step i the sale s_i equals the order amount (i.e., demand is censored), assume that the input demand d_i is the largest possible M.

Note assuming that when demand is censored $d_i = M$ does not affect the immediate working or profit of $\mathcal{C}$INC. It only affects the decisions $\mathcal{C}$INC takes in future.

Theorem 3. $\mathcal{C}\mathsf{INC}(\mathsf{cen}(\sigma)) \geq \mathsf{STOPT}(\sigma) - T\Delta \max\{r-c, c\}$.

Proof. Let σ' be the input sequence $\mathcal{C}$INC assumes it receives—after the censorship and its own modifications on censored demands. Every demand value in σ' is at least as large at the corresponding values in σ. Thus $\mathsf{STOPT}(\sigma') \geq \mathsf{STOPT}(\sigma)$. The rest follows from Lemma 2.

We can similarly define $\mathcal{C}$INCβ and prove the following theorem.

Theorem 4. $\mathcal{C}\mathsf{INC}\beta(\mathsf{cen}(\sigma)) \geq \mathsf{STOPT}(\sigma) - ((T+t)\delta \max\{r-c, c\} + O(|X - x|^2 r/\delta))$, *in which* T *and* t *are the lengths of* σ *and* β, *respectively, and* δ *is the* Δ*-value for* (β, σ).

4 Experiments

4.1 Comparison Algorithms

In our experiments, we implement and compare our algorithms with the following:

STOPT. This is the static optimal described in Section 1. At every step it orders $d^{(\lceil T - Tc/r \rceil)}$, the demand at position $\lceil T - Tc/r \rceil$ in the sequence of demands sorted in non-decreasing order, as shown in [4].

NORMAL. This algorithm assumes that the demands are normally distributed with a given mean μ and standard deviation σ, and places the order that maximizes the expected profit. That order amount is $\mu + \sigma\phi^{-1}\big((r-c)/r\big)$, where $\phi^{-1}(\cdot)$ is the inverse of the standard normal cumulative distribution function [12].

SCARF. This algorithm implements Scarf's Rule [41]. Rather then using the normal distribution, like above, it assumes a worst case distribution for the given mean μ and standard deviation σ. It orders $\mu + \frac{\sigma}{2}\big(\sqrt{(r-c)/c} - \sqrt{c/(r-c)}\big)$ if $c(1 + \sigma^2/\mu^2) < r$, and 0 otherwise [12, 41].

WMN *and* WMNS. These algorithms, Weighted Majority Newsvendor and Weighted Majority Newsvendor Shifting, respectively, are based on the weighted majority approach that is common in computational learning theory [31, 32, 33]. They are the first known algorithms with worst-case performance bounds.

CAVE. This algorithm, named Concave, Adaptive Value Estimation, estimates the the profit as a function of the order amount, through a sequence of concave, piecewise linear approximations. The approximations are constructed using sample gradients of the profit at different order quantities [14]. It does not depend on any prior knowledge of the underlying sample distribution. Furthermore, it is designed to handle censored demands.

Algorithms WMN, WMNS, and CAVE use user-specified parameters. For those, we use values recommended in their respective papers.

4.2 Experiments on Simulated Data

First, in Figure 1(a) we show the performance of INC on a simulated demand sequence generated from a normal distribution. We took the average profit of 100 sequences of 100 days (steps) each. The actual means of these sequences were about 52 and the standard deviations were about 10. The minimum value for each sequence was 0 and the maximum value was 100. Further, we chose revenue per item $r = 5$ and cost per item $c = 2$. (These values for r and c are used for all experiments, including the ones on real-world data.) We compare our basic INC to STOPT, SCARF, NORMAL, WMN, and WMNS. The demand sequence was not censored, as these algorithms are designed for non-censored data. Along the x-axis we have the various values of the mean assumed by SCARF and NORMAL. Along the y-axis we have the profit of each algorithm.

Notice that INC is extremely close to WMN and WMNS all of three of which are slightly less then STOPT. They outperform NORMAL and SCARF when the

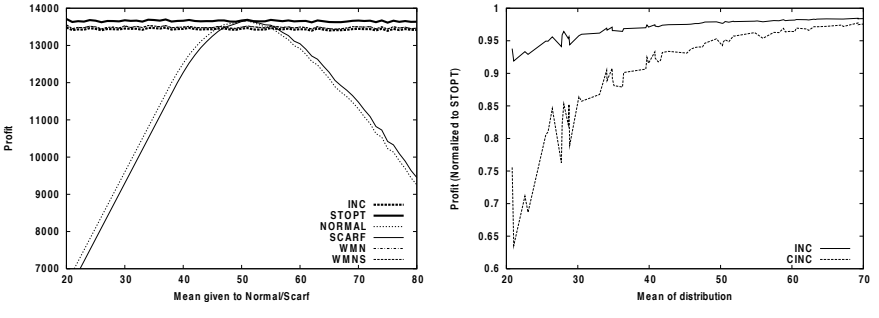


Fig. 1. (a) INC is close to WMN and WMNS, and outperforms NORMAL and SCARF when they assume means off by about 3% or more. (b) Assuming data is censored reduces profit in the basic $\mathcal{C}$INC.

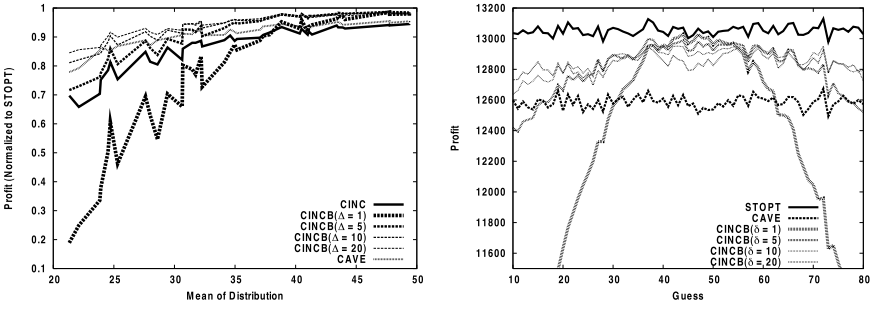


Fig. 2. (a) Bounding Δ in $\mathcal{C}$INCβ improves performance, as long as β is short. (b) $\mathcal{C}$INCβ is tolerant to bad guesses of X.

assumed mean is off by about just 3 or more (which translates to about 3% of the range or more).

For the remaining tests, we focus on performance on censored demand sequences. In Figure 1(b), we again generate simulated values from different normal distributions, such that the mean demand values of the sequences vary from 20 to 70. We compare the performance of the basic INC algorithm to that of its censored version $\mathcal{C}$INC. The mean of the sequences vary along the x-axis, and the profits of the algorithms as a fraction of the profits of STOPT are shown on the y-axis.

For data sets in which the mean is significantly less then the maximum value of demand, there is a severe drop-off of performance of our censored algorithm $\mathcal{C}$INC. This drop can be accounted for by noting that the bound on the INC and $\mathcal{C}$INC, without added β sequences, is in terms of Δ, the maximum "gap" between the demands. When we are using the censored data approach Δ can be quite large since several demand values are assumed to be M even when the sequence has only seen much smaller demands. In other words, $\mathcal{C}$INC often ends up choosing M as the order amount, even when most of the demands are much smaller. This produces considerable waste.

The above problem is addressed by adding an initial set of inputs in the algorithm $\mathcal{C}$INCβ, effectively bounding Δ to some chosen small value δ. We show results of $\mathcal{C}$INCβ's performance in Figure 2(a). Here we attempted several versions of $\mathcal{C}$INCβ, but without any form of guess X. That is, we simply added a string of prior demands a specified gap δ apart. Versions of $\mathcal{C}$INCβ with Δ bounded by 5, 10, or 20, all perform better than $\mathcal{C}$INC. The version of $\mathcal{C}$INCβ with Δ bounded by 1 does not, and this is because the corresponding β is quite long and adversely affects the performance on the relatively short σ. (This is later countered by adding a suitable guess for X to $\mathcal{C}$INCβ.)

Observe that $\mathcal{C}$INCβ, with equally spaced prior gaps, seems to perform best if the sequence's mean was close to 55. Since at this value it turns out that the

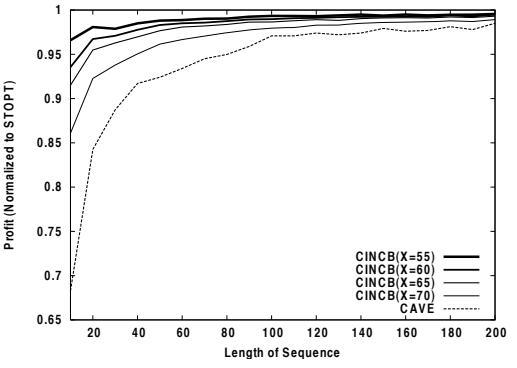


Fig. 3. The influence of beginning sequence β diminishes with increasing length of input sequence σ

amount x that STOPT orders on the sequence is close to the amount STOPT orders on the β sequences we chose, it led us to the idea of guessing described in Section 3.3. Instead of letting β influence the final result, we can control the effect with a guess X for x, and creating a β such that STOPT on β would order X. In Figure 2(b) we show the performance of this idea. This data is also simulated, and has an actual mean around 47. The sequence length is 100. On the x-axis we have the values of the guesses given to the algorithms. There are several versions of $\mathcal{C}$INCβ, each with different gap values, $\Delta = \delta$.

Recall from Section 3.4 that the bound on $\mathcal{C}$INCβ is: $\mathcal{C}\mathsf{INC}\beta(\mathsf{cen}(\sigma)) \geq \mathsf{STOPT}(\sigma) - ((T+t)\delta \max\{r-c, c\} + O(|X-x|^2 r/\delta))$. This equation predicts that the amount of regret resulting from a bad guess is quadratic in the incorrectness of the guess. And, our graphs of $\mathcal{C}$INCβ do actually have a parabolic shape. Further, notice that the curves with wider gaps slope downward in profit more slowly, as the guesses become worse—they are robust to bad guesses. However, small gap-versions have a benefit if the guess is accurate: their peaks achieve higher profits. We also plotted CAVE in this plot. Cave does reasonably well, and has the advantage of requiring no initial guess. However, for guesses within 15 units of the optimal, giving a window of about 35 out of a range of 100, all $\mathcal{C}$INCβ versions outperform CAVE. The $\mathcal{C}$INCβ versions with δ of 10 or 20 outperform CAVE irrespective of the guess.

The quadratic term within the bound equation for $\mathcal{C}$INCβ above does not have a T factor within in it. That is, it is independent of the length of the sequence. As such, we expect that for longer sequences, the amount of error, when taken as a percentage of STOPT, will decrease—as the length of the sequence σ dominates over that of β. We show this in Figure 3. The four plots of $\mathcal{C}$INCβ are for various guesses for X. The y-axis for this (and remaining plots) shows the profit as a fraction of the profit earned by STOPT. The sequences are generated so that the optimal guess would be 55. The plots are clearly as expected.

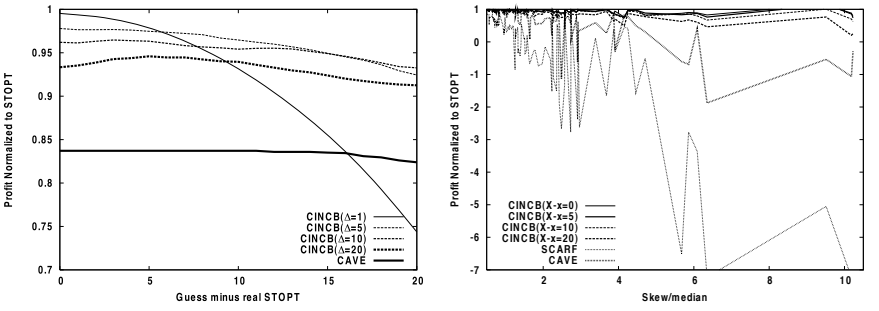


Fig. 4. (a) $\mathcal{C}$INCβ performs well on real-world data as well. (b) $\mathcal{C}$INCβ's performance is steady even as the relative skew in real-world increases.

4.3 Experiments on Real-World Data

We also tested our algorithms using real-world data from a local supermarket. We took the amount of sales for 250 dairy products for a sequence of about 260 days, and used them as our demand sequences. Here too, for simplicity, we used revenue per item $r = 5$ and cost per item $c = 2$. This data is very different then the normal distributions that we used earlier—many sequences have large skew.

In Figure 4(a), we take the same algorithms as from Figure 2(b). The x-axis is a normalized guess value. That is, the value on the x-axis is the difference between the guess X and the optimal guess x (STOPT choice for σ). Since all of the demand sequences are quite different, we use this normalization. The value plotted against the y-axis is the average profit over the 250 products, normalized using STOPT's profit for each product. Notice that the relationships in Figure 2(b) still hold for the $\mathcal{C}$INCβ algorithms on real-world data. Smaller gaps are stronger at the peak, but slope away faster.

From the last plot, we find that CAVE averages a lower performance then before. We believe this is due to the non-normality in real-world data. We use the value of the skew of each distribution (the difference between median and mean), as a measure of the non-normality. Figure 4(b) has, on the x-axis, the skew of each real world sequence, normalized by the median (i.e., $|\text{mean} - \text{median}|/\text{median}$). Relatively more skewed distributions appear farther right. Each plot of $\mathcal{C}$INCβ has a gap Δ of 5. The four plots show four different variations from the optimal x of the guessed value X. We also plot the performance of SCARF and CAVE. Here we use a version of SCARF which takes the demands from the sequence seen till now and computes their mean and standard deviation, which it uses in Scarf's Rule to compute the next order quantity. The demand values given to SCARF are uncensored.

In this graph, we see that the bounds on $\mathcal{C}$INCβ hold even for highly erratic data. While CAVE performs very well so long as the sequence has small relative skew, the other sequences result in poor performance, even *net losses* over the entire sequence. SCARF is much worse off on real data, even though it has the advantage of uncensored demands.

5 Conclusion

In introducing our incremental algorithm and proving its bounds, we introduce an algorithm with three strengths. First, the algorithm is fairly simple and we can directly bound its profit in terms of the profit of the offline optimal algorithm STOPT (the only known such bound for the newsvendor problem). Second, the simplicity of our algorithm implies that it can be modified to account even for censored data. Finally, the algorithm's performance is robust on both simulated data and real-world data.

It would be interesting to extend the algorithms to more general versions of the newsvendor problem. For instance, when the items ordered in a period do not become obsolete at the end of that very period, but can be carried over for a fixed number of periods.

References

1. Al-Binali, S.: A Risk-Reward Framework for the Competitive Analysis of Financial Games. Algorithmica 25(1), 99–115 (1999)
2. Azoury, K.: Bayes solution to dynamic inventory models under unknown demand distribution. Management Science, 1150–1160 (1985)
3. Bergemann, D., Schlag, K.: Robust monopoly pricing: The case of regret. Working Paper (2005)
4. Bertsimas, D., Thiele, A.: A data driven approach to newsvendor problems. Technical report, Massechusetts Institute of Technology, Cambridge, MA (2005)
5. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
6. Burnetas, A., Smith, C.: Adaptive ordering and pricing for perishable products. Operations Research 48(3), 436–443 (2000)
7. Cesa-Bianchi, N., Lugosi, G.: Prediction, Learning, and Games. Cambridge University Press, Cambridge (2006)
8. Chamberlain, G.: Econometrics and decision theory. Journal of Econometrics 95(2), 255–283 (2000)
9. Chen, L., Plambeck, E.: Dynamic inventory management with learning about the demand distribution and substitution probability. Manufacturing & Service Operations Management 10(2), 236 (2008)
10. Conrad, S.: Sales data and the estimation of demand. Operational Research Quarterly, 123–127 (1976)
11. Ding, X., Puterman, M., Bisi, A.: The censored newsvendor and the optimal acquisition of information. Operations Research, 517–527 (2002)
12. Gallego, G.: Ieor 4000: Production management lecture notes, http://www.columbia.edu/~gmg2/4000/pdf/lect_07.pdf
13. Gallego, G., Moon, I.: The distribution free newsboy problem: Review and extensions. Journal of the Operational Research Society 44(8), 825–834 (1993)
14. Godfrey, G.A., Powell, W.B.: An adaptive, distribution-free algorithm for the newsvendor problem with censored demands, with applications to inventory and distribution. Management Science 47(8), 1101–1112 (2001)
15. Harpaz, G., Lee, W., Winkler, R.: Learning, experimentation, and the optimal output decisions of a competitive firm. Management Science, 589–603 (1982)

16. Huh, W., Janakiraman, G., Muckstadt, J., Rusmevichientong, P.: Asymptotic optimality of order-up-to policies in lost sales inventory systems. Management Science 55(3), 404–420 (2009)
17. Huh, W., Rusmevichientong, P.: An Asymptotic Analysis of Inventory Planning with Censored Demand. Technical report, Columbia Working Paper (2006)
18. Iglehart, D.: The dynamic inventory problem with unknown demand distribution. Management Science, 429–440 (1964)
19. Kalai, A., Vempala, S.: Efficient algorithms for online decision problems. Journal of Computer and System Sciences 71(3), 291–307 (2005)
20. Kalai, A., Vempala, S.: Efficient algorithms for online decision problems. J. Comput. Syst. Sci. 71(3), 291–307 (2005)
21. Karlin, S.: Dynamic inventory policy with varying stochastic demands. Management Science, 231–258 (1960)
22. Lariviere, M., Porteus, E.: Stalking information: Bayesian inventory management with unobserved lost sales. Management Science, 346–363 (1999)
23. Levi, R., Roundy, R., Shmoys, D.: Provably near-optimal sampling-based algorithms for stochastic inventory control models. In: Proc. ACM Symposium on Theory of computing, pp. 739–748. ACM, New York (2006)
24. Lim, A.E.B., Shanthikumar, J.G.: Relative entropy, exponential utility, and robust dynamic pricing. Operations Research 55(2), 198–214 (2007)
25. Littlestone, N., Warmuth, M.K.: The weighted majority algorithm. Information and Computation, 212–261 (1994)
26. Liyanage, L., Shanthikumar, J.: A practical inventory control policy using operational statistics. Operations Research Letters 33(4), 341–348 (2005)
27. Lu, X., Song, J., Zhu, K.: Inventory control with unobservable lost sales and Bayesian updates. Working Paper (2005)
28. Lu, X., Song, J., Zhu, K.: Analysis of Perishable-Inventory Systems with Censored Demand Data. Operations Research 56(4), 1034 (2008)
29. Murray Jr., G., Silver, E.: A Bayesian analysis of the style goods inventory problem. Management Science, 785–797 (1966)
30. Nahmias, S.: Demand estimation in lost sales inventory systems. Naval Research Logistics 41(6) (1994)
31. O'Neil, S.: Online learning for the newsvendor problem. Master's thesis, University of Notre Dame (2009)
32. O'Neil, S., Chaudhary, A.: Comparing online learning algorithms to stochastic approaches for the Multi-Period newsvendor problem. In: Workshop on Algorithm Engineering and Experiments, p. 49 (2008)
33. O'Neil, S., Zhao, X., Sun, D., Chaudhary, A., Wei, J.C.: Coping with demand shocks: A distribution-free algorithm for solving newsvendor problems with limited demand information. (under review)
34. Perakis, G., Roels, G.: Regret in the newsvendor model with partial information. Operations Research 56(1), 188–203 (2008)
35. Porteus, E.L.: Foundations of Stochastic Inventory Theory. Stanford University Press, Stanford (2002)
36. Powell, W., Ruszczyński, A., Topaloglu, H.: Learning algorithms for separable approximations of discrete stochastic optimization problems. Mathematics of Operations Research, 814–836 (2004)
37. Raman, A., Fisher, M.: Reducing the cost of demand uncertainty through accurate response to early sales. Operations Research 44(4), 87–99 (1996)
38. Savage, L.J.: The theory of statistical decisions. Journal of the American Statistical Association 46, 55–67 (1951)

39. Scarf, H.: Bayes solutions of the statistical inventory problem. The Annals of Mathematical Statistics, 490–508 (1959)
40. Scarf, H.: Some remarks on Bayes solutions to the inventory problem. Naval Research Logistics Quarterly 7(4), 591–596 (1960)
41. Scarf, H.E.: A min-max solution of an inventory problem. Stanford University Press, Stanford (1958)
42. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM 32(3), 652–686 (1985)
43. Takimoto, E., Warmuth, M.K.: Path kernels and multiplicative updates. Journal of Machine Learning Research 4, 773–818 (2003)
44. Vairaktarakis, C.L.: Robust multi-item newsboy models with a budget constraint. International Journal of Production Economics, 213–226 (2000)
45. Weisstein, E.: Order statistic. From MathWorld—A Wolfram Web Resource, http://mathworld.wolfram.com/OrderStatistic.html
46. Yu, G.: Robust economic order quantity models. European Journal of Operations Research 100(3), 482–493 (1997)
47. Zhang, G., Xu, Y.: A Risk-Reward Competitive Analysis for the Newsboy Problem with Range Information. In: Du, D.-Z., Hu, X., Pardalos, P.M. (eds.) COCOA 2009. LNCS, vol. 55, p. 345. Springer, Heidelberg (2009)

$O((\log n)^2)$ Time Online Approximation Schemes for Bin Packing and Subset Sum Problems*

Liang Ding[1], Bin Fu[1], Yunhui Fu[1], Zaixin Lu[1], and Zhiyu Zhao[2]

[1] Department of Computer Science, University of Texas-Pan American, Edinburg, TX 78539, USA
dliang@broncs.utpa.edu, binfu@cs.panam.edu, fuyunhui@gmail.com, lzaixin@broncs.utpa.edu

[2] Department of Computer Science,University of New Orleans, New Orleans, LA 70148, USA
sylvia@cs.uno.edu

Abstract. Given a set $S = \{b_1, \cdots, b_n\}$ of integers and an integer s, the subset sum problem is to decide if there is a subset S' of S such that the sum of elements in S' is exactly equal to s. We present an online approximation scheme for this problem. It updates in $O(\log n)$ time and gives a $(1+\epsilon)$-approximation solution in $O((\log n + \frac{1}{\epsilon^2}(\log \frac{1}{\epsilon})^{O(1)}) \log n)$ time. The online approximation for target s is to find a subset of the items that have been received. The bin packing problem is to find the minimum number of bins of size one to pack a list of items $a_1, \cdots, a_n$ of size in $[0, 1]$. Let function $\mathrm{bp}(L)$ be the minimum number of bins to pack all items in the list L. We present an online approximate algorithm for the function $\mathrm{bp}(L)$ in the bin packing problem, where L is the list of the items that have been received. It updates in $O(\log n)$ updating time and gives a $(1+\epsilon)$-approximation solution $\mathrm{app}(L)$ for $\mathrm{bp}(L)$ in $O((\log n)^2 + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})})$ time to satisfy $\mathrm{app}(L) \leq (1+\epsilon)\mathrm{bp}(L) + 1$.

1 Introduction

Both the subset sum and the bin packing are classical problems that have been studied extensively in the area of theoretical computer science. An instance of the subset sum problem is a pair (S, s), where S is a set $\{b_1, \cdots, b_n\}$ and s is an integer. The target is to find if there is a subset S' of S such that the sum of elements in S' is exactly equal to s. Ibarra and Kim [12] have given a fully polynomial-time approximation for it. The fastest approximation scheme was shown by Kellerer et al [15] with running time $O(\min\{n/\epsilon, n + (1/\epsilon)^2(\log(1/\epsilon)\})$.

The bin packing problem is to find the minimum number of bins of size one to pack a list of items $a_1, \cdots, a_n$ of size in $[0, 1]$. It is a classical NP-hard problem and has been widely studied. The bin packing problem has many applications in the engineering and information sciences. Some approximation algorithms have

* This research is supported by NSF Career Award 0845376.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 250–261, 2010.

been developed for this problem. Examples include First Fit, Best Fit, Sum-of-Squares, and Gilmore-Gomory cuts [2,8,7,11,10]. The first linear time approximation scheme for the offline bin packing problem is shown in [9]. Recently, a sublinear time approximation algorithm has been developed for the offline bin packing problem with weighted sampling [3]. A classical online algorithm assigns items to bins in the order they are given in the original list, without using the information of subsequent items. Online algorithms and their performance were reported in a series of papers [13,14,18,6,16,19,21,20]. The current champion online algorithm has approximation ratio 1.58889 given by Seiden [20]. On the other hand, a lower bound of 1.53635 for online algorithms has been proved by Brown [4] and Liang [17].

For online models of computation, an item from input can arrive at any time, and will be saved in the memory. For the subset sum problem, when a target s is given, a $(1+\epsilon)$-approximation solution should be outputted. For the bin packing problem, a $(1+\epsilon)$-approximation solution should be outputted for packing all items that have arrived.

We show an online approximation scheme for the subset sum problem such that it has a $O(\log n)$ updating time and gives a $(1+\epsilon)$-approximation solution in an $O((\log n + \frac{1}{\epsilon^2}(\log \frac{1}{\epsilon})^{O(1)}) \log n)$ time. The online approximation for target s is to find a subset of the items that have been received. We also show an $O((\log n + \frac{1}{\epsilon^2}(\log \frac{1}{\epsilon})^{O(1)}) \log n)$ time approximation scheme for the subset sum problem with a sorted list of items.

Let function $\mathrm{bp}(L)$ be the minimum number of bins to pack all items in the list L. We present an online approximate algorithm for the function $\mathrm{bp}(L)$ in the bin packing problem. It updates in $O(\log n)$ time and gives a $(1+\epsilon)$-approximation solution $\mathrm{app}(L)$ for $\mathrm{bp}(L)$ in $O((\log n)^2 + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})})$ time to have $\mathrm{app}(L) \leq (1+\epsilon)\mathrm{bp}(L) + 1$, where L is the list of input items received. The online algorithms in this paper for bin packing only approximate the minimum number of bins to pack those input items. It also gives a packing plan that allows an item position to be changed at different moment. This does not contradict the existing lower bound works [4,17] that no approximation scheme for online algorithms that do not change the bins of already packed items.

Our algorithms for two problems share some similar technologies, which are based on partitioning the input items into $O(\log n)$ intervals. In section 2, we present algorithms and their lower bound for the subset sum problem. In section 3, we present algorithms and their lower bound for the bin packing problem.

1.1 Overview of Our Methods

For a target s, the elements are partitioned as large items and small items to stay in $O(\log n)$ intervals $[a_1, a_2), [a_2, a_3), \cdots, [a_k, a_{k+1}]$. A large item is of size at least δs for some constant δ. In each interval, there is a small factor $(1+\gamma)$ difference among its elements for some small positive constant γ. If the elements are saved in a tree structure like the B-tree, the number of input items in each interval can be determined in $O(\log n)$ steps. For the subset sum problem, apply a pruning method to approximate the sum of subsets with large items to reduce

the number of cases. Finally, add some small items from those intervals to make the total sum approximation to the target as close as possible.

For the bin packing problem, a linear programming method is used to pack large items [9]. Then small items are first filled into the bins packed with items, and then some additional bins. The number of items in each interval shows how much capacity is needed.

2 Algorithms for the Subset Sum Problem

Given a list of positive integers $b_1, b_2, \cdots, b_n$ and a positive integer s, the subset sum problem is to find a subset of elements from the list such that their sum is equal to s. The *optimal solution* to this problem is the subset with the largest sum, but no more than s. In this section, we study an approximation to this problem, which tries to minimize the difference with the optimal solution.

2.1 An Offline Algorithm for Subset Sum

We show an algorithm that depends on certain data structure to hold the input items and a pruning method. We present a pruning method in Section 2.2 and a data structure in Section 2.3.

Definition 1.
- *A multiset has the format* $(n_1 g_1, \cdots, n_m g_m)$*, where each* $n_i g_i$ *represents that element* g_i *appears* n_i *times. The number* m *is called the length of the multiset.*
- *For a multiset* $T = (n_1 g_1, \cdots, n_m g_m)$*, define* $\sum(T) = n_1 g_1 + \cdots + n_m g_m$.
- *Assume that* T *is a multiset* $(n_1 g_1, \cdots, n_m g_m)$*. A sub-multiset* T' *of* T *is a multiset* $(n'_1 g_1, \cdots, n'_m g_m)$ *with* $n'_i \leq n_i$ *for* $i = 1, \cdots, m$.
- *For a multiset* $T = (n_1 g_1, \cdots, n_m g_m)$ *and a real* $\gamma > 0$*, a* γ*-pruning of* T *is a list of integers* $s_1, \cdots, s_t$ *such that for each sub-multiset* $T' = (n'_1 g_1, \cdots, n'_m g_m)$ *of* T*, there is a* s_i *such that* $\sum(T')/(1+\gamma) \leq s_i \leq \sum(T')$ *and* $s_i = \sum(T'')$ *for some sub-multiset* $T'' = (n''_1 g_1, \cdots, n''_m g_m)$ *of* T.
- *A multiset* $T = (n_1 g_1, \cdots, n_m g_m)$ *is a* (δ, m, m')*-multiset for* s *if for each* g_i $(i = 1, \cdots, m)$*,* $\delta s \leq g_i \leq s$ *and* $n_i \leq m'$.

In the algorithm below, we assume that we have an algorithm $Prune(\gamma, T)$ that returns a γ-pruning for a multi-set T.

Algorithm

Input: a parameter ϵ, which determines the ratio of approximation, an integer target s, and a data structure to hold the input integers $b_1, \cdots, b_n$.

Steps:

1. Let constants $\epsilon_1 = \frac{\epsilon}{100}$ and $\gamma = \frac{\epsilon_1}{100}$.
2. Find the largest element b_{k_1} that is at most s.
3. Partition all items less than or equal to b_{k_1} into $O(\log n)$ intervals $I_1 = [a_1, a_0), I_2 = [a_2, a_1), \cdots, I_t = [a_t, a_{t-1}]$, where $a_0 = b_{k_1}$, and $a_t \leq \frac{\gamma s}{n}, a_{i+1} = a_i/(1+\gamma)$.

4. Let the first group of intervals consist $I_1, I_2, \cdots, I_m$ such that $a_m \leq \epsilon_1 s$ and $a_{m-1} > \epsilon_1 s$.
5. Let the second group of intervals consist $I_{m+1}, I_{m+2}, \cdots, I_t$.
6. Let $M = Prune(\gamma, (\min(\lfloor \frac{2}{\epsilon_1} \rfloor, C[I_1])a_1, \cdots, \min(\lfloor \frac{2}{\epsilon_1} \rfloor, C[I_m])a_m))$, where $C[I_i]$ is the number of items in I_i, and $\lfloor \frac{2}{\epsilon_1} \rfloor$ is for the consideration that a subset with sum at most s should have no more than $\lfloor \frac{2}{\epsilon_1} \rfloor$ items of size at least $\epsilon_1 s$.
7. Let $M' = \emptyset$.
8. For each $y \in M$ with $y \leq s$,
9. Let $v_1 = y$.
10. For each $I_i = [a_i, a_{i-1})$ in the second group,
11. Find the largest integer $j_i \leq C[I_i]$ with $v_1 + j_i \cdot a_i \leq s$
12. Let $v_1 = v_1 + j_i \cdot a_i$.
13. Put v_1 into set M'.
14. Output the largest $x \in M'$.

End of Algorithm

Lemma 1. *Assume that there is an algorithm $Prune(\gamma, T)$ such that given a multiset (δ, m, m')-multiset T for s, it generates a γ-pruning in $O(t(\frac{1}{\epsilon}))$ time if $m = O(\frac{1}{\epsilon})$, $m' = O(\frac{1}{\epsilon})$ and $\delta = O(\epsilon)$. Assume that the n input elements are saved in the data structure D that takes $q(n)$ time to answer the number of items in an interval range $[a, b]$ and the largest item at most s. Then there is a deterministic $O((q(n) + t(\frac{1}{\epsilon})) \log n)$ time algorithm such that it outputs an $(1+\epsilon)$-approximation for the subset sum problem, where ϵ is an arbitrary positive constant.*

Proof. Without loss of generality, assume $\epsilon \leq 1$. Consider an optimal solution $s_{opt} = u_1 + u_2 = (b_{i_1} + b_{i_2} + \cdots + b_{i_h}) + (b_{j_1} + b_{j_2} + \cdots + b_{j_k})$ for the subset sum problem, where $u_1 = b_{i_1} + b_{i_2} + \cdots + b_{i_h}$ is the sum of items from the first group of intervals and $u_2 = b_{j_1} + b_{j_2} + \cdots + b_{j_k}$ is the sum of items from the second group of intervals. There exists a $u_1' \in M$ with

$$\frac{u_1}{1+\gamma} \leq u_1' \leq u_1. \tag{1}$$

Convert u_2 into $u_2' = a_{j_1} + a_{j_2} + \cdots + a_{j_k}$ such that $b_{j_u} \in I_{j_u} = [a_{j_u}, a_{j_u - 1})$ for $u = 1, \cdots, k$. Therefore,

$$\frac{u_2}{1+\gamma} \leq u_2' \leq u_2. \tag{2}$$

We have $s_{opt}/(1+\gamma) \leq u_1' + u_2'$.

According to lines 10 to 13 in the algorithm, there exists $v_1 \in M'$ such that

$$v_1 = u_1' + u_2'' = u_1' + (j_{m+1}a_{m+1} + j_{m+2}a_{m+2} + \cdots + j_t a_t), \tag{3}$$

where

$$u_2'' = (j_{m+1}a_{m+1} + j_{m+2}a_{m+2} + \cdots + j_t a_t). \tag{4}$$

Since each item in the second group is at most $\epsilon_1 s$ and our algorithm increases v_1 as much as possible until it is in $[s - \epsilon_1 s, s]$ or all items in the second group is used, we have

$$v_1 \geq min(u_1' + u_2', (1 - \epsilon_1)s). \tag{5}$$

Case 1. The optimal solution $s_{opt} \leq (1 - \epsilon_1)s$. By inequality (5), we have

$$\begin{aligned} v_1 &\geq \min(u_1' + u_2', (1 - \epsilon_1)s) && (6)\\ &\geq \min(u_1' + u_2', s_{opt}) \geq \min(s_{opt}/(1 + \gamma), s_{opt}) && (7)\\ &\geq s_{opt}/(1 + \gamma). && (8) \end{aligned}$$

– Case 1.1. $s_{opt}(1 + \epsilon_1) < v_1$.
For each term $j_p a_p$ in u_2'' (see equation (4)), find an arbitrary subset $J_p \subseteq I_p$ such that J_p has exactly j_p elements. Let $u_2''' = \sum_{p=m+1}^{t}(\sum_{b_i \in J_p} b_i)$, and $v_2 = u_1 + u_2'''$. We have

$$u_2'' \leq u_2''' \leq (1 + \gamma)u_2''. \tag{9}$$

By inequalities (1) and (9), we have $v_1 \leq v_2 \leq v_1(1 + \gamma)$.
  - Case 1.1.1 $v_2 > s$.
  We know that $u_1 \leq s$ since $s_{opt} = u_1 + u_2 \leq s$. We reduce some items of u_2''' from the second type of intervals to convert v_2 to v_3. We will eventually have $(1-\epsilon_1)s \leq v_3 \leq s$ since each item from the second group of intervals is at most $\epsilon_1 s$. Therefore, $s_{opt} < v_3 \leq s$, which contradicts that s_{opt} is an optimal solution.
  - Case 1.1.2 $v_2 \leq s$.
  Since $s_{opt} < v_1 \leq v_2 \leq s$, this contradicts that s_{opt} is an optimal solution.

– Case 1.2. $s_{opt}(1 + \epsilon_1) \geq v_1$.
By inequalities (6) to (8), we have $\frac{s_{opt}}{1+\gamma} \leq v_1$. Thus, we have $\frac{s_{opt}}{1+\gamma} \leq v_1 \leq s_{opt}(1 + \epsilon_1)$. Therefore, v_1 is an $(1 + \epsilon)$-approximation to s_{opt}.

Case 2. The optimal solution $s_{opt} > (1 - \epsilon_1)s$.
By inequalities (1) and (2), $\frac{u_1+u_2}{1+\gamma} \leq u_1' + u_2' \leq u_1 + u_2$. By lines 10 to 13 in the algorithm and equation (3), we have $v_1 = u_1' + u_2'' \geq u_1' + u_2' - \epsilon_1 s$ since each item in a second type interval is at most $\epsilon_1 s$. Therefore,

$$\frac{s_{opt}}{1 - \epsilon_1} \geq s \geq v_1 = u_1' + u_2'' \geq u_1' + u_2' - \epsilon_1 s \geq \frac{u_1 + u_2}{1 + \gamma} - \epsilon_1 s \tag{10}$$

$$\geq \frac{u_1 + u_2}{1 + \gamma} - \epsilon_1 \frac{s_{opt}}{(1 - \epsilon_1)} = (\frac{1}{1 + \gamma} - \frac{\epsilon_1}{(1 - \epsilon_1)})s_{opt} \geq (1 - \frac{\epsilon}{2})s_{opt}. \tag{11}$$

Therefore, $v_1 \in [(1 - \frac{\epsilon}{2})s_{opt}, \frac{s_{opt}}{1-\epsilon_1}]$. Therefore, v_1 is an $(1 + \epsilon)$-approximation for s_{opt}.

It takes $O(q(n)\log n)$ time to obtain $C[I_i]$ for $i = 1, \cdots, t$ since $t = O(\log n)$. The number of first type intervals is $O(\log \frac{1}{\epsilon})$. Each approximate solution subset for target s contains at most $O(\frac{1}{\epsilon})$ items from first type intervals. It takes $O(t(\frac{1}{\epsilon})\log n)$ time to prune and process all $y \in M$ in lines 8 to 13 in the algorithm. The total time is $O((q(n) + t(\frac{1}{\epsilon}))\log n)$. □

2.2 A Pruning Method

We show a pruning method that is embedded in the algorithm in Section 2.1. It is based on a divide-and-conquer method.

Consider a (δ, m, m')-multiset $T = (n_1 b_1, \cdots, n_u b_u)$ for s for $\delta = O(\epsilon), m = O(\frac{1}{\epsilon})$, $m' = O(\frac{1}{\epsilon})$. The following functions are used to generate a δ'-pruning for T for $\delta' = O(\epsilon)$.

The parameter δ' be used to control the pruning. Let $\eta = \frac{\delta'}{\log u}$. Partition $[\delta s, 2s]$ into intervals $J_1 = [d_0, d_1), J_2 = [d_1, d_2), \cdots, J_v = [d_{v-1}, d_v]$ such that $d_i = d_{i-1}(1+\eta)$. We just make the number v to be big enough such that $(1+\eta)^v \geq 2s/\delta s = 2/\delta$. Therefore, the total number of intervals $v = O(\frac{(\log u)(\log \frac{1}{\epsilon})}{\epsilon})$ by the setting of η and the fact $\delta' = O(\epsilon)$.

Sketch(M)
Input: a list of items $M = h_1, \cdots, h_r$
    Let $U = \emptyset$.
    For each interval J_i, put the smallest item in $J_i \cap M$ into U.
    Output U.
End of Sketch

Merge (M_1, M_2)
Input: two lists of items $M_1 = x_1, \cdots, x_t$, and $M_2 = y_1, \cdots, y_z$
    Let $M = \emptyset$.
    For each pair $x_i \in M_1$ and $y_j \in M_2$
        put $x_i + y_j$ into M.
    Sketch(M).
End of Merge

Algorithm Prune(δ', L)
Input: A multiset $L = n_1 b_1, \cdots, n_u b_u$ and a parameter δ'.
    Let $M = \emptyset$.
    If L has only one item, $n_1 b_1$
        Put $0, b_1, 2b_1, \cdots, n_1 b_1$ into list M.
        Return Sketch(M).
    Partition L into two multisets L_1 and L_2 evenly.
    Return Merge(Prune(L_1), Prune(L_2)).
End of Algorithm

The following facts are easy to verify. Their proofs can be found in [5].

Lemma 2 ([5]). *(1)For $0 \leq y \leq 1$, $e^y \leq 1+y+y^2$. (2)For a real number $y \geq 1$, $(1+\frac{1}{y})^y \leq e$. (3) For a real number y, $1+y < e^y$.*

Lemma 3. *There is an $O((\frac{1}{\epsilon})^2(\log(\frac{1}{\epsilon}))^{O(1)})$ time algorithm such that given a (δ, m, m')-multiset for s, where $\delta = O(\epsilon), m = O(\frac{1}{\epsilon})$, $m' = O(\frac{1}{\epsilon})$, it generates a δ'-pruning for a $\delta' = O(\epsilon)$.*

Proof. We prove this by induction. Assume $u = 2^k$ for some integer k, where u is the number of different elements in multiset T. When $2^{k-1} < u < 2^k$ for some integer k, we can append some 0s so that the total length is equal to 2^k.

The basis of induction is trivial. Assume that for $u \leq 2^{k-1}$, for each multiset L of length u, and each sub-multiset L' of L, Prune(δ', L) contains h with $\sum(L')/(1+\eta)^{k-1} \leq h \leq \sum(L')$. Consider the case $u = 2^k$. Let L be a multiset of length $u = 2^k$. Partition L into L_1 and L_2 of length 2^{k-1}. Let L' be a sub-multiset with partition $L' = L'_1 \cup L'_2$, where L'_1 consists all elements in L_1 and L'_2 consists all elements in L_2. By our hypothesis, we have $h_1 \in$ Prune(δ', L'_1) and $h_2 \in$ Prune(δ', L'_2) such that $\sum(L'_1)/(1+\eta)^{k-1} \leq h_1 \leq \sum(L'_1)$ and $\sum(L'_2)/(1+\eta)^{k-1} \leq h_2 \leq \sum(L'_2)$. By the merging and pruning procedures, we have item h with $(h_1+h_2)/(1+\eta) \leq h \leq h_1+h_2$, which implies $\sum(L')/(1+\eta)^k \leq h \leq \sum(L')$. Therefore, for the input multiset $L' = (n_1 b_1, \cdots, n_u b_u)$ for s, $\sum(L')/(1+\eta)^{\log u} \leq h \leq \sum(S')$. By Lemma 2,

$$h \geq \sum(L')/(1+\eta)^{\log u} \geq \sum(L')e^{-\eta \log u} \tag{12}$$

$$\geq \sum(L')e^{-\delta'} \geq \sum(L')/(1+\delta'+\delta'^2) \tag{13}$$

$$\geq \sum(L')/(1+2\delta'). \tag{14}$$

The total number of intervals $v = O(\frac{(\log u)(\log \frac{1}{\epsilon})}{\epsilon})$ by the setting of η and the fact $\delta' = O(\epsilon)$. It is easy to see that the merging takes $O((\frac{\log u}{\epsilon})^2)$ each time. The total time follows from the recursive equation $T(u) \leq 2T(u/2) + O((\frac{(\log u)(\log \frac{1}{\epsilon})}{\epsilon})^2)$. This gives a solution $T(u) = O((\frac{(\log u)(\log \frac{1}{\epsilon})}{\epsilon})^2 \log u) = O((\frac{1}{\epsilon})^2(\log \frac{1}{\epsilon})^2(\log u)^3) = O((\frac{1}{\epsilon})^2(\log \frac{1}{\epsilon})^5)$ since $u = O(\frac{1}{\epsilon})$. □

2.3 A Data Structure

We describe a data structure to hold the input elements. It takes $O(\log n)$ time to insert a new element and find the number of elements in the range of an arbitrary interval. It is an interval 2-3 tree [1,5].

Definition 2.
- *A 2-3 tree [1,5] is a tree whose each internal node has two or three children, and every path from the root to a leaf is of the same length.*
- *Define an interval 2-3 tree to be a 2-3 tree such that each internal node A has an additional record for the number of leaves in the subtree with root at A.*

– *For an interval 2-3 tree T and a node A of T, let T_A be the subtree of T with root at A.*

Lemma 4. *An interval 2-3 tree takes $O(\log n)$ time to insert a new element and $O(\log n)$ time to obtain the number of elements in a prescribed range $[a, b]$.*

Proof. Assume T is an interval 2-3 tree to hold all elements from the input. Each internal node A also holds the number of leaves in the subtree T_A. This makes it possible to find the number of items in a range $[a, b]$ in $O(\log n)$ steps. Insertion to the interval 2-3 tree is similar to the regular 2-3 tree, but it also increase the size of subtrees that contain the new item.

All items in the interval 2-3 tree are in the bottom leaves with an ascending order from left to right. In order to find the number of items in a prescribed range $[a, b]$, search the interval 2-3 tree for the leftmost item a' that is larger than or equal to a, and also the rightmost item b' that is less than or equal to b. Let $P_{a'}$ be the path from a' to the root, and $P_{b'}$ be the path from b' to the root. Let r' be the first node that is in both $P_{a'}$ and $P_{b'}$. Following the path from a' to r', calculate the number n_1 of leaves on the left side of a' (the leaves less than a') under subtree $T_{r'}$. Following the path from b' to r', calculate the number n_2 of leaves on the right side of b' (the leaves greater than b') under subtree $T_{r'}$. Let n_0 be the size of subtree $T_{r'}$. Then $n_0 - n_1 - n_2$ is the number of leaves in the range $[a, b]$. □

A node contains the tuple $(l_1, \cdots, l_k, s)$, where k is the number of children, l_i is the largest leaf under the substree of the i-th child, and s is the number of leaves under this node. Furthermore, a node also holds the tuple $(p_1, \cdots, p_k)$, where each p_i is a pointer to the i-th child.

2.4 Full Algorithms for the Subset Sum Problem

We show both online and offline algorithms for the subset sum problem. For the online algorithm, a new items a_i or target s arrives at unpredicted time. When a target is given, an $(1+\epsilon)$ approximation of the optimal solution will be computed. We also convert our online an algorithm that deals with the subset sum problem with multiple targets and have Theorem 3.

Theorem 1. *There is an online algorithm for the subset sum problem such that it has an $O(\log n)$ updating time, and computes an $(1+\epsilon)$-approximation in $O((\log n + \frac{1}{\epsilon^2}(\log \frac{1}{\epsilon})^{O(1)}) \log n)$ time, where ϵ is an arbitrary positive constant.*

Proof. Build an interval 2-3 tree as described in Lemma 4. When a new item arrives, insert it in the interval 2-3 tree. It follows from Lemma 1 and Lemma 4. □

The following theorem is for an offline algorithm with a sorted list of input.

Theorem 2. *There is a $O((\log n + \frac{1}{\epsilon^2}(\log \frac{1}{\epsilon})^{O(1)}) \log n)$ time algorithm such that given a list of n sorted elements and a target s, it outputs a $(1+\epsilon)$-approximation for the subset sum problem, where ϵ is an arbitrary positive constant.*

Proof. It is well known that searching a sorted list takes $O(\log n)$ steps with binary search. It follows from Lemma 3 and Lemma 1. □

Definition 3. *Given a list of positive integers $a_1, \cdots, a_n$, and m targets $t_1, \cdots, t_m$, the approximate multiple queries subset sum problem is to find $(1+\epsilon)$-approximation solutions for the m targets.*

Theorem 3. *There is an algorithm for the multiple queries subset sum problem such that it outputs m $(1+\epsilon)$-approximations in $O((n + m(\log n + (\frac{1}{\epsilon})^2(\log \frac{1}{\epsilon})^{O(1)}))\log n)$ time, where ϵ is an arbitrary positive constant.*

Proof. It follows from Theorem 1. □

2.5 Lower Bound for Subset Sum

We give an $\Omega(\log n)$ lower bound for deterministic approximation scheme for the subset sum problem with a sorted input list.

Theorem 4. *Every deterministic approximation scheme must make $\Omega(\log n)$ adaptive queries to the input sorted list for the subset sum problem.*

3 Algorithms for Bin Packing

In this section, we first show an $O(\log n)$ time deterministic approximation scheme for bin packing if the input is a sorted list of elements. We then show an online approxiamtion scheme which takes $O(\log n)$ time and $O(n)$ space.

3.1 Bin Packing for Large Items

In this section, we review a classical method for packing large items.

Definition 4.
- *For item y and integer h, define y^h to be h copies of item y.*
- *A type T_i of a bin is represented by a multi-set $(b_{1,i}a_1, \cdots, b_{m,i}a_m)$, which satisfies $\sum_{j=1}^{m} b_{j,i}a_j \leq 1$.*
- *If T is a type of bin, denote (x, T) to be x bins of type T.*
- *A packing solution is given by a list of $(x_1, T_1), \cdots, (x_t, T_t)$ to pack all elements.*
- *Assume that $y_1 \leq y_2 \leq \cdots \leq y_m$ be a sublist of items in the list $L = a_1 \cdots a_n$. Assume that P is a packing solution for $L' = y_1^{h_1} \leq y_2^{h_2} \leq \cdots \leq y_m^{h_m}$. Define a packing adaption of L' to L is a packing that put h_i elements between y_{i-1} and y_i to replace those slots for y_i.*
- *A packing scheme is often described as an adaption to an existing solution for some list $y_1^{h_1} \leq y_2^{h_2} \leq \cdots \leq y_m^{h_m}$.*

We show the following Lemma 5 and Lemma 6 that are essentially from [9].

Lemma 5 ([9]). *Assume δ is a constant. Given a bin packing problem for $B = \{n_1 a_1, \cdots, n_m a_m\}$ with each $a_i \geq \delta$, there is a $m^{O(\frac{1}{\delta})}$ time algorithm to give a solution with at most $Opt(B) + q$, where q is the number of types to pack $a_1, \cdots, a_m$, and is at most $m^{\frac{1}{\delta}}$.*

Lemma 6 ([9]). *Assume there is a $t(m, n)$ time algorithm A such that given a list of items of size at least δ, it returns m items $y_1, y_2, \cdots, y_m$, where y_i is the i-th element from the list for $i = 1, 2, \cdots, m$. Then there is a $t(O(1), n) + (\frac{1}{\epsilon\delta})^{O(\frac{1}{\delta})}$ time approximation scheme B for the δ-bin packing problem.*

3.2 An Offline Algorithm

Our offline algorithm for the bin packing problem assumes that the input is a sorted list of items. Our algorithm takes $O((\log n)^2)$ time. It is interesting that this gives an example of an NP-hard problem that has an $O((\log n)^2)$ time approximation scheme.

Lemma 7. *Assume that the n input elements are saved in the data structure D that takes $y(n)$ time to answer the number of items in the range $[a, b]$ or find the i-th element. Then there is a deterministic $O((\log n)y(n) + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})})$ time algorithm such that it outputs a $(1 + \epsilon)C_{opt} + 1$ approximation for the bin packing problem, where ϵ is an arbitrary positive constant.*

Proof. Without loss of generality, assume $0 < \epsilon \leq 1$. Let constants $\delta = \frac{\epsilon}{3}$ and $\gamma = \frac{\epsilon}{3}$.

Algorithm

1. Partition all items in $[\frac{\delta}{n^2}, \delta)$ into $O(\log n)$ intervals $I_1 = [a_0, a_1), I_2 = [a_1, a_2), \cdots, I_t = [a_{t-1}, a_t)$, where $a_0 = \frac{\delta}{n^2}$, $a_t = \delta, a_{i+1} = (1 + \gamma)a_i$.
2. Let $C(I_i)$ be the number of items in the interval I_i.
3. Use $s_i = C([a_i, a_{i+1}))a_{i+1}$ to approximate the sum of items in $[a_i, a_{i+1})$.
4. Find a $(1+\epsilon)$-approximation U of bins for packing all items of size at least δ via Lemma 6 and let $x_1 T_1, \cdots, x_q T_q$ be the types from the above approximate solution.
5. Fill all bins in $x_1 T_1, \cdots, x_q T_q$ with at least δ space unfilled by the items in $\cup_{i=1}^{t} C([a_i, a_{i+1}))a_{i+1}$ so that each bin wastes no more than δ space.
6. Use some additional bins to pack the left items in $\cup_{i=1}^{t} C([a_i, a_{i+1}))a_{i+1}$.
7. Make the plan that each item in $[a_{i-1}, a_i]$ replaces an a_{i+1} packed in those bins (the plan is not executed).

End of Algorithm

By the algorithm, we have $\frac{1}{1+\gamma} \sum_{j=1}^{t} s_j \leq \sum_{a_i \in [\frac{1}{n^2}, \delta)} a_i \leq \sum_{j=1}^{t} s_j$. Assume that an optimal solution to a bin packing problem has two types of bins. Each bin of the first type contains at least one item of size δ, and each bin of the second type only contains items of size less than δ. Let R_1 be the set of all first type bins, and R_2 be the set of all second type bins.

Case 1. If U can contain all items, then $|U| \leq (1+\epsilon)|R_1| \leq (1+\epsilon)|R_1 \cup R_2|$.

Case 2. A bin beyond those in U is used. Let U' be all bins without more than δ positions left right after line 6 in the algorithm. There is at most one bin with more than δ space wasted right after line 6. In line 7 of the algorithm, when an item a_{i+1} is replaced by an item in $[a_i, a_{i+1})$, at most $a_{i+1}\frac{\gamma}{1+\gamma}$ additional space is wasted. After line 7, a bin in U' wastes at most $\delta + (1-\delta)\frac{\gamma}{1+\gamma} \leq 2\delta$ space. We have $|U'| \leq \frac{|R_1 \cup R_2|}{(1-2\delta)} \leq (1+\epsilon)|R_1 \cup R_2|$. Therefore, the approximate solution is at most $(1+\epsilon)|R_1 \cup R_2| + 1$.

It takes $O((\log n)y(n))$ time to generate $s_i = C([a_i, a_{i+1}))a_{i+1}$ for $i = 1, \cdots, t$. It takes $O(y(n) + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})})$ time to derive the approximation for those elements in $[\delta, 1]$ by Lemma 5. The total time is $O((\log n)y(n) + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})})$. □

Theorem 5. *There is a deterministic $O((\log n)^2 + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})})$ time algorithm such that given a list of n sorted elements, it outputs a $(1+\epsilon)C_{opt}+1$ approximation to the bin packing problem, where ϵ is an arbitrary positive constant.*

Proof. It follows from Lemma 7 and the fact that it takes $O(\log n)$ time find the number of elements in a prescribed range $[a, b]$. □

3.3 An Online Algorithm

We show a deterministic online algorithm to approximate the bin packing problem. We use a balance tree to store items from the input list. The tree supports insertion and searching in $O(\log n)$ time. The method used in Theorem 5 will be applied here.

Theorem 6. *There is a deterministic streaming algorithm for the bin packing problem such that it has a $O(\log n)$ updating time, and computes a $(1+\epsilon)C_{opt}+1$ approximation in $((\log n)^2 + (\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}$ time, where ϵ is an arbitrary positive constant.*

Proof. Build an interval 2-3 tree as described in Lemma 4. When a new item arrives, insert it in the interval 2-3 tree. It follows from Lemma 7 and Lemma 4. □

3.4 Lower Bound for Deterministic Algorithms

Theorem 7. *Every deterministic approximation scheme must make $\Omega(\log \log n)$ adaptive queries to a sorted input list for the bin packing problem.*

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Applegate, D., Buriol, L., Dillard, B., Johnson, D., Shore, P.: The cutting-stock approach to bin packing: Theory and experiments. In: Proceedings of Algorithm Engineering and Experimentation (ALENEX), pp. 1–15 (2003)

3. Batu, T., Berenbrink, P., Sohler, C.: A sublinear-time approximation scheme for bin packing. Theoretical Computer Science 410, 5082–5092 (2009)
4. Brown, D.: A lower bound for on-line one-dimensional bin packing problem. Technical Report 864, University of Illinois, Urbana, IL (1979)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press, Cambridge (2001)
6. Csirik, J., Imreh, B.: On the worst-case performance of the nkf bin-packing heuristic. Acta Cybern. 9(2), 89–90 (1989)
7. Csirik, J., Johnson, D., Kenyon, C., Orlin, J., Shore, P., Weber, R.: A self organizing bin packing heuristic. In: Goodrich, M.T., McGeoch, C.C. (eds.) ALENEX 1999. LNCS, vol. 1619, pp. 246–265. Springer, Heidelberg (1999)
8. Csirik, J., Johnson, D., Kenyon, C., Orlin, J., Shore, P., Weber, R.: On the sum-of-squares algorithm for bin-packing. In: Proceedings of the 22nd annual ACM symposium on theory of computing (STOC), pp. 208–217 (2000)
9. Fernandez de la Vega, W., Lueker, G.S.: Bin packing can be solved within 1+epsilon in linear time. Combinatorica 1(4), 349–355 (1981)
10. Gilmore, M., Gomory, R.: A linear programming approach to the cutting-stock problem-part ii. Operations Research 11(6), 863–888 (1963)
11. Gilmore, M., Johnson, D.: A linear programming approach to the cutting-stock problem. Operations Research 29(6), 1094–1104 (1981)
12. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. J. ACM 22(4), 463–468 (1975)
13. Johnson, D.: Near-Optimal Bin Packing Algorithms. PhD thesis, Massachusets Institute of Technology, Department of Computer Science, Cambridge (1973)
14. Johnson, D.S.: Fast algorithms for bin packing. Journal of Computer and System Sciences 8(3), 272–314 (1974)
15. Kellerer, H., Mansini, R., Pferschy, U., Speranza, M.: An efficient fully polynomial approximation scheme for the subset-sum problem. Journal of Computer and System Sciences 66(2), 349–370 (2003)
16. Lee, C.C., Lee, D.T.: A simple on-line bin-packing algorithm. Journal of ACM 32(3), 562–572 (1985)
17. Liang, F.: A lower bound for on-line bin packing. Information processing letters 10, 76–79 (1980)
18. Mao, W.: Tight worst-case performance bounds for next-k-fit bin packing. SIAM J. Comput. 22(1), 46–56 (1993)
19. Ramanan, P., Brown, D.J., Lee, C.C., Lee, D.T.: On-line bin packing in linear time. J. Algorithms 10(3), 305–326 (1989)
20. Seiden, S.S.: On the online bin packing problem. Journal of ACM 49(5), 640–671 (2002)
21. Yao, A.: New algorithms for bin packing. Journal of ACM 27(2), 207–227 (1980)

Path Separability of Graphs

Emilie Diot and Cyril Gavoille*

Université de Bordeaux, LaBRI, France
{diot,gavoille}@labri.fr

Abstract. In this paper we investigate the structural properties of k-path separable graphs, that are the graphs that can be separated by a set of k shortest paths. We identify several graph families having such path separability, and we show that this property is closed under minor taking. In particular we establish a list of forbidden minors for 1-path separable graphs.

1 Introduction

"Divide and Conquer" is an common technique in computer science to solve problems. The whole data is separated into different small parts to find a solution in these parts, and then to merge the solutions to obtain the result on the input graph.

A wide theory has been developed for graphs that can be decomposed into small pieces. Such graphs, a.k.a. bounded treewidth graphs, supports polynomial algorithms for many class of problems, whereas no algorithms of complexity better than exponential complexity are known for the general case. This has contributed to new insights into Fixed Parameterized Tractable theory whose consequences for practical algorithms are effective improvements on the running time [2,6].

There are however problems that can be efficient solved even for graphs without small separators (or equivalently of large treewidth). Large separators but "well shaped" reveal very useful for approximation algorithms. For instance, if the separator has a small diameter, or a small dominating set, then distances between vertices can be computed efficiently up to some small additive errors (see [4,3,12] for works about the treelength of a graph).

An important observation due to Thorup [11] is that separators consisting of a set of shortest paths are also very useful for the design of compact routing scheme, distance and reachability oracles. More precisely, he used the fact that every weighted planar graph with n vertices has a set of three shortest paths whose deletion split the graph into connected components of at most $n/2$ vertices (a decomposition into components of at most $2n/3$ vertices using two shortest paths was early proved in [7]). Using a recursive decomposition, and sampling

* The second author is also member of the "Institut Universitaire de France". Both authors are supported by the ANR-project "ALADDIN", and the équipe-projet commune LaBRI-INRIA Bordeaux Sud-Ouest "CÉPAGE".

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 262–273, 2010.

each such shortest path, he showed that distances between any pair of vertices can be approximated up to a factor of $1+\varepsilon$ in polylogarithmic time, for every $\varepsilon > 0$, with an oracle of size[1] $O(\varepsilon^{-1} n \log n)$.

This notion of "shortest path" separator has been extended in [1]. Roughly speaking, a k-path separator is the union of k shortest paths whose removal halve the graph. The formal definition is actually slightly more complicated and is described in Section 2. The same authors have showed that k-path separable graphs have efficient solutions for several "Object Location Problems" including compact routing schemes, distance oracles, and small-world navigability. Based on the deep Robserton-Seymour's decomposition [9], they show in particular that every weighted graph excluding a minor H has a $(1+\varepsilon)$-approximate distance oracle of size $O(\varepsilon^{-1} kn \log n)$, where $k = k(H)$ depending only of H. Actually, the oracle can be distributed into balanced labels, each of size $O(\varepsilon^{-1} k \log n)$ such that distance queries can be answered from given the source-destination labels only. The graphs excluding a fixed minor is a huge family of graphs including (and not restricted to) bounded treewidth graphs, planar graphs, and graphs of bounded genus.

1.1 Our Results

An approximate distance oracle for a graph G is a data-structure that quickly returns, for any source-destination pair, an approximation on cost of a shortest path connecting them. Such data-structures are obtained by preprocessing G where each edge has a weight corresponding to the cost of traversing this edge (or length). However, in practice, the number n of vertices of G is large whereas the number of interesting vertices for which we want approximate the distance is small (say t). Current solutions [1,11] provide oracles of size $O(t \log n)$ whereas a space independent of n would be preferable.

Such a compression can be achieved by adding weights on the vertices of the input graph. Typically, interesting vertices receive a weight 1 whereas the others receive a weight 0. A k-path separator on such vertex- and edge-weighted graph is then defined as previously, excepted that the removal of the separator must leave connected components of size at most half the total vertex-weight of the graph. The size of the oracles is improved since $\log t$ recursion levels suffice instead of $\log n$ in the initial formulation.

In this paper we extend the classical notion of k-path separability to edge- and vertex-weighted graphs. In particular, we prove that previous results (e.g., the 3-path separability of planar graphs) still hold in this new framework.

We establish a connection between separators corresponding to the border of a face and k-path separability, and we identify several families of graphs that are 1-path and 2-path separable. We note that most of our proofs are constructive, and lead to polynomial and even linear algorithms.

More interestingly, we show that the family of graphs that are k-path separable for any weight function is minor-closed. Combined with the Graph Minor Theory

[1] The *size* is actually the number of "distance items" stored in the oracle.

of Robertson and Seymour [10], it follows that the k-path separability can be theoretically tested in cubic time [8] for each fixed k, although no algorithm is currently known. Finally, we provide a first step towards the characterization of 1-path separable graphs.

2 Preliminaries

A *minor* of a graph G is a subgraph of a graph obtained from G by edge contraction. We denote by K_r the complete graph (or clique) on r vertices, and $K_{p,q}$ the complete bipartite graph. For convenience, the term *component* is a short for connected component.

A *vertex-weight function* (resp. *edge-weight function*) is a non-negative real function defined on the vertices (resp. edges) of a graph. A non-negative real function applying on both vertices and edges is simply called *weight function*. A weighted graph is graph G having a weight function ω, that we denote also by (G, ω). The *weight* of a subgraph H of G, denoted by $\omega(H)$, is the sum of the weights over the vertices of H.

A *half-separator* for a graph G with vertex-weight function ω is a subset of vertices S such that each component of $G \setminus S$ has weight at most $\omega(G)/2$. Observe that the deletion of a half-separator does not necessarily disconnect the graph.

A *k-path separator* of a weighted graph G is a subgraph $P_0 \cup P_1 \cup \ldots$ where each P_i is a subgraph composed of the union of k_i minimum cost paths in $G \setminus \bigcup_{j<i} P_j$, and where $\sum_i k_i \leqslant k$. A k-path separator is said *strong* if it consists of P_0 only, i.e., composed of the union of k minimum cost paths in G. A weighted graph is (strongly) k-path separable if every induced subgraph has a (strong) k-path separator.

A *tree-decomposition* of a graph G is a tree T whose vertices, called *bags*, are subsets of vertices of G such that:

1. for every vertex u of G, there exists a bag X of T such that $u \in X$;
2. for every edge $\{u, v\}$ of G, there exists a bag X of T such that $u, v \in X$; and
3. for every vertex u of G, the set of bags containing u induces a subtree of T.

An important property following from the last two points is that every path between $u \in X$ and $v \in Y$ in G has to intersect all the bags on the path from X and Y in T. Therefore, the deletion of every bag X disconnects G provided that $T \setminus X$ is composed of more than one subtree and that no bags $Y \subseteq X$.

The *width* of a tree-decomposition T is $\max_{X \in T} \{|X| - 1\}$. A *treewidth-$t$* graph is a graph having a tree-decomposition of width t, and the treewidth of G is the minimum t such that G is a treewidth-t graph.

We will use several times the following basic result (the proof of this lemma, and of several others, appear in the full version).

Lemma 1. *Every tree-decomposition of a vertex-weighted graph has a bag that is a half-separator of the graph. Such a bag is called* center *of the tree-decomposition.*

For the sake of presentation, we prove the following folklore results.

Proposition 1

1. *Every weighted treewidth-t graph is strongly $\lceil (t+1)/2 \rceil$-path separable.*
2. *Every weighted planar graph is strongly 3-path separable.*
3. *Every weighted n-vertex graph is strongly $\lceil n/4 \rceil$-path separable*
4. *The uniform*[2] *weighted clique K_{4k+1} is not k-path separable.*

Proof.

1. Consider any subgraph H of a weighted graph G. The treewidth of H is at most the treewidth of G. So H has a tree-decomposition of width $\leqslant t$. By Lemma 1, the center C of the tree-decomposition is a half-separator. It can be covered by at most $\lceil |C|/2 \rceil \leqslant \lceil (t+1)/2 \rceil$ shortest paths. Therefore, H has a strong $\lceil (t+1)/2 \rceil$-path separator, and thus G is strongly $\lceil (t+1)/2 \rceil$-path separable.

2. It is well-known that every planar graph has a tree-decomposition such that every bag consists of at most three shortest paths. This comes from the well-known fact that every planar graph having a depth-h rooted tree has a tree-decomposition where each bag consists of 3 paths of the tree starting from the root (see [5][pp. 305]). By Lemma 1, the center C of the tree-decomposition is a half-separator. So, C forms a strong 3-path separator.

3. Consider any subgraph H of a weighted graph (G, ω) with n vertices. Let W be the smallest set of vertices in H such that $\omega(W) \geqslant \omega(H)/2$. Thus, the components of $H \setminus W$ have weight $\leqslant \omega(H)/2$. It is clear that W contains at most half the vertices of H, i.e., $|W| \leqslant \lceil |V(H)|/2 \rceil$. A set of at most $\lceil |W|/2 \rceil$ shortest paths suffices to cover W. Therefore, H has a strong k-path separator with $k = \lceil \lceil |V(H)|/2 \rceil /2 \rceil \leqslant \lceil n/4 \rceil$, completing the proof of Point 3.

4. Let us show that the uniform weighted K_{4k+1} has no k-path separator. Indeed, every k-path separator S consists of at most $2k$ vertices since every shortest path in a clique consists of an edge. $K_{4k+1} \setminus S$ is a clique on at least $2k+1$ vertices, so S is not a half-separator. □

As remark in [1], the k-path separability of minor-free graphs holds also for vertex-weighted graphs. However the formal proof of this result cannot be considered as folklore, and its self-contained proof is currently more than the page limitation of this paper.

One the of unresolved problem we left open is to know whether they are planar graphs that are not 2-path separable.

3 Face-Separable Graphs

As we will see later in Section 4, graphs that are k-path separable have strong structural properties. In particular, planarity plays an important role, at least for $k = 1$ in the light of Theorem 3. In this section we will see that, a half-separator

[2] That is with a unit weight for all vertices and edges.

of a special "shape" implies a low path separability of the graph. Moreover, this half-separator is defined independently of the shortest path metric of the graph, it only depends on the vertex-weight function.

A half-separator S of a weighted graph G is a *face-separator* if G has a plane embedding such that S is the border of a face. A graph is *face-separable* if every induced subgraph has a face-separator.

By definition, outerplanar graphs are face-separable, since the outerface contains all vertices of the graph. We will see that the family of face-separable graphs includes more general graphs, like the series-parallel graphs, the subdivisions of a K_4 (Proposition 2), and even includes some unbounded treewidth planar graphs (Proposition 3).

The main result of this section is:

Theorem 1. *Every face-separable weighted graph is strongly* 2*-path separable.*

The bound given by Theorem 1 is best possible because there are face-separable graphs that are not 1-path separable. This can be proved by combining Proposition 2 and the fact there are treewidth-2 graphs and subdivisions of K_4 that are not 1-path separable – see Fig. 3.

Proposition 2. *Every weighted treewidth-*2 *graph or weighted subdivision of* K_4 *is face-separable.*

Proof. Let (G, ω) be any weighted treewidth-2 graph. It is known that every treewidth-2 graph is a subgraph of a series-parallel graph, and in particular a planar graph. As any subgraph of G is also a treewidth-2 graph, it is sufficient to prove that G has a face-separable. We consider the graph H obtained from G by adding as many edges as possible while preserving a treewidth-2 graph. Let T be a tree-decomposition of H of width 2, and let C be the center of T. Bag C is composed of a K_3. We embed H in the plane such that C is the border of a face of this embedding. This is possible by moving some subgraph from inside the K_3 to outside. If not, H would contain a K_4-minor, contradicting the fact that H has treewidth 2. We can now remove the edges that have been added to H in order to obtain G, and we consider the border S of the face containing the three vertices of C. Such a face exists since deleting edges can only enlarge the existing faces of a plane embedding. We have $S \subseteq C$, and C is a half-separator of H (Lemma 1). Note that H has the same total weight of G (we have added only edges). It follows that S is a half-separator for G. This completes the first part of the statement of the proposition.

Consider now a subdivision G of K_4 having a vertex-weight function ω, and H be an induced subgraph of G. If H is a proper subgraph of G (i.e., $H \neq G$), then H is outerplanar and thus has a face-separator. So, assume that $H = G$.

We assume given a plane embedding of H. We denote by $v_1, \ldots, v_4$ the four degree-3 vertices of H, and by $P_{i,j}$ the path between v_i and v_j, for all $i, j \in \{1, \ldots, 4\}$. Let $w_i = \omega(v_i)$, and let $p_{i,j} = \omega(P_{i,j} \setminus \{v_i, v_j\})$ be the sum of the weights of vertices on $P_{i,j}$ excluding its extremities.

Let assume that H has no face-separator. There are four possible faces $F_1, \ldots, F_4$, each one bordered by three paths. Faces are ordered such that whenever the border of F_i is removed, the remaining component is composed of three paths sharing vertex v_i. The total weight of this component is $w_i + \sum_{j \neq i} p_{i,j}$. As the border of F_i is not a face-separator, we have $w_i + \sum_{j \neq i} p_{i,j} > \omega(H)/2$. This holds for each $i \in \{1, \ldots, 4\}$. Summing these four equations, it turns out (observe that each path $p_{i,j}$ occurs twice in this sum):

$$\sum_i w_i + 2 \sum_{i \neq j} p_{i,j} \;>\; 2\omega(H) = 2 \left(\sum_i w_i + \sum_{i \neq j} p_{i,j} \right) .$$

It implies $0 > \sum_i w_i$: a contradiction, by definition $\omega(v) \geqslant 0$ for each vertex v. Therefore, one of the face F_i is a face-separator for H, that completes the proof. □

Up to now, the graphs we have proved to be face-separable are all of treewidth $\leqslant 3$. From Proposition 1 (Point 1), they are 2-path separable. It is natural to ask whether all face-separable graphs have such a low treewidth property. We answer negatively to this question.

Proposition 3. *For every n, there is a uniform weighted face-separable graph with at most n vertices whose treewidth is $\Omega(\log \log n)$.*

Proof. (Sketch). The proof is based on the construction of a graph called G_p, for integral $p \geqslant 1$. It has treewidth at least $k = p - O(\log \log p)$ because we can show it contains a $k \times k$-grid minor, and the number of vertices of G_p is $n < 2^{2^p}$. In other words, the treewidth of G_p is at least $\log \log n - O(\log^{(4)} n)$.

Graph G_p is composed of a tree T_p of depth p where each vertex of depth $i < p$ has exactly $d(i)$ children, for some function d defined later. Furthermore, for each depth i, a path linking all depth-i vertices is added to T_p to form G_p. Let us denote by $L(i)$ the number of depth-i vertices in T_p. The values $L(i)$ and $d(i)$ obey to the following induction: $L(0) = 1$ and $L(i) = L(i-1) \cdot d(i-1)$, where $d(i) = \sum_{j=0}^{i} L(j)$. The first values of $L(i)$ and $d(i)$ are given in the table hereafter, and G_4 is depicted on Fig. 1.

i	0	1	2	3	4	5	...
L	1	1	2	8	96	10368	...
d	1	2	4	12	108	10464	...

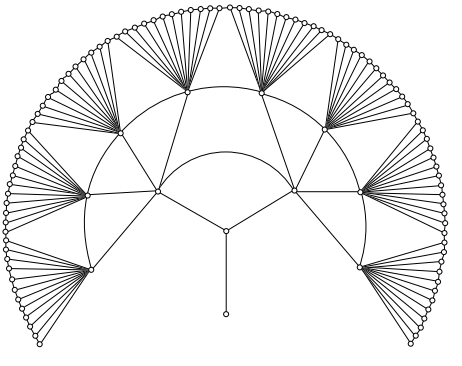


Fig. 1. The graph G_4 with 108 vertices

To prove Proposition 3, we show that every subgraph H of G_p contains a face-separator. An important property we use is that in G_p, the number $d(i)$ of children for a vertex of depth i is at least the vertex number of the graph induced by T_{i-1}. The key point is that H is either outerplanar, or there must exist a vertex v of depth i in T_p such that all its children belongs to the border of the outerface of H. In the first case, H is trivially face-separable. In the second one, and using the property on $d(i)$, we derive that at least half the vertices of H lie on the outerface. □

4 The Hierarchy of Separable Graphs

For every integer $k \geqslant 1$, we denote by PS_k the family of all the graphs G that are k-path separable for every weight function ω. More formally,

$$\mathrm{PS}_k \;=\; \{G \mid \forall \omega, (G,\omega) \text{ is } k\text{-path separable}\}\ .$$

We define similarly the family SPS_k of all the graphs that are strongly k-path separable for every weight function.

We have seen that every weighted planar graph is strongly 3-path separable. In other words, planar graphs are in SPS_3. In Section 3, we have seen that every subdivided outerplanar graph is face-separable, and thus strongly 2-path separable. Thus this family is in SPS_2. We will show in Proposition 6 that outerplanar graphs are in actually in PS_1. Obviously, families PS_1 and SPS_1 coincide.

Clearly, for each k, $\mathrm{SPS}_k \subset \mathrm{PS}_k$ since a strongly k-path separator is a particular k-path separator. Also, the hierarchies $\mathrm{PS}_1 \subset \cdots \subset \mathrm{PS}_k$ and $\mathrm{SPS}_1 \subset \cdots \subset \mathrm{SPS}_k$ are strict because of the complete graph. By Proposition 1 (points 3 and 4), $K_{4k+1} \in \mathrm{SPS}_{k+1}$ and $K_{4k+1} \notin \mathrm{PS}_k$. The family PS_k is however much larger than SPS_k as suggested by the next proposition.

Proposition 4. *For each $k > 4$, there is a graph G_k with $O(k^2)$ vertices such that $G_k \notin \mathrm{SPS}_k$, but $G_k \in \mathrm{PS}_4$.*

Proof. Consider the graph G_k composed of a $2(k+1) \times 2(k+1)$-grid in which a vertex v is connected to all the vertices of the grid. G_k has $4k^2+5$ vertices. Vertex and edge weights are unitary. As shown in [1], G_k has no strong k-path separator, the removal of any set of k shortest paths deletes at most $2k+1$ vertices, and $2(k+1)$ are required to halve the graph. Therefore, $G_k \notin \mathrm{SPS}_k$. However, for every weight function ω, (G_k, ω) has a 4-path separator. The first path consists of the universal vertex v, and the three others are defined as in the planar case (since $G_k \setminus \{v\}$ is planar, and thus 3-path separable). Therefore, $G_k \in \mathrm{PS}_4$. □

For the study of PS_k and SPS_k graphs families, the next proposition tell us that we can always assume that graphs are biconnected.

Proposition 5. *A graph belongs to PS_k (reps. SPS_k) if and only if all its bicomponents belong to PS_k (resp. SPS_k).*

4.1 Closed under Minor Taking

The remarkable property of PS_k and SPS_k families is they are closed under minor taking. From the Graph Minor Theorem, such families can be characterized by a finite list of forbidden minors, and membership of a given graph in one of these families can be done in time $O(n^3)$ for fixed k.

Theorem 2. *For each integer $k \geqslant 1$, the families PS_k and SPS_k are minor-closed.*

Proof. We give the proof for the family SPS_k, the proof for PS_k is similar. Let H be any minor of a graph G. We will prove that if G is k-path separable, then H is k-path separable too. To prove that H is k-path separable, we need to prove the property for every induced subgraph of H. However, since every subgraph of H is also a minor of G, we simply show that H has a k-path separator.

It is not difficult to see that if H is a minor of G, then with each vertex u of H we can associate a connected subgraph of G, called *super-node* of u, such that if (u, v) is an edge of H, then there exists an edge of G, called *super-edge* of (u, v), connecting a vertex of the super-node of u and a vertex of the super-node of v. (If there are several such edges we select only one.) The super-nodes must be pairwise disjoint (see Fig. 2).

Let ω_H be any weight function on H. From ω_H, we construct a weight function ω_G on G as follows. For every edge (x, y) of G that is a super-edge of (u, v) (colored black on Fig. 2), we set $\omega_G(x, y) = \omega_H(u, v)$. For every edge (x, y) of G such that x and y both belongs to the same super-node (called *internal-edge* and dashed on Fig. 2), we set $\omega_G(x, y) = 0$. And, for all other edges (x, y) of G (called *external-edge* and colored red on Fig. 2), we set $\omega_G(x, y) = 1 + \sum_{e \in E(H)} \omega_H(e)$, so that the cost of a path in G using any such edge is strictly larger than the cost of any simple path in H. The weight of a vertex x that belongs to the super-node of u is $\omega_G(x) = \omega_H(u)/t_u$, where t_u is the number of vertices of the super-node of

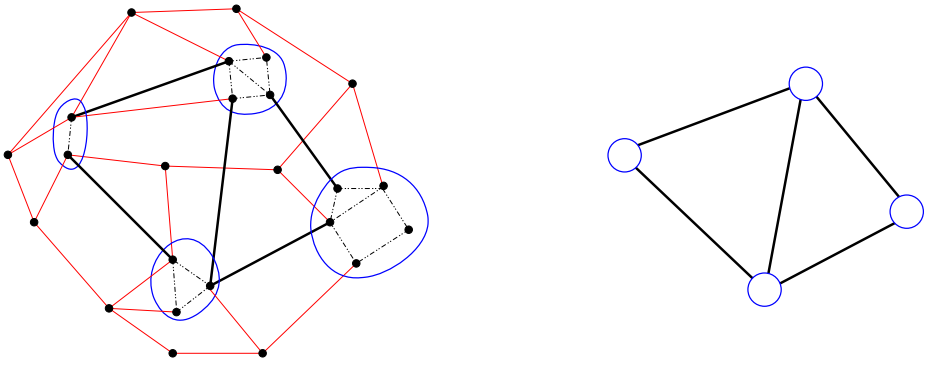


Fig. 2. A graph G and a minor H

u. Note that the sum of weights of the vertices of the super-node of u is precisely $\omega_H(u)$. The weight of all other vertices is 0. Observe that $\omega_G(G) = \omega_H(H)$.

Since $G \in \mathrm{SPS}_k$, the weighted graph (G, ω_G) has a k-path separator S_G consisting of k shortest paths in G. Let $H_0, H_1, \ldots$ be the components of H, and assume that $\omega_H(H_0)$ is maximum. With each path P of S_G that intersects a super-node of a vertex of H_0, we associate a path Q in H_0 as follows. Let $(U_0, \ldots, U_t)$ be the ordered sequence of all the super-nodes of vertices of H_0 traversed by P. We denote by u_i the vertex of H_0 such U_i is the super-node of u_i. Path Q is obtained by adding an edge between u_{i-1} to u_i, for each $i \in \{1, \ldots, t\}$. We claim that the set composed of each path Q constructed from P as above, and denote by S_H, is a half-separator of H.

First, let us show that Q is a shortest path in H_0 (and thus in H). Path P between the last vertex of U_{i-1} and the first vertex of U_i consists of the super-edge of (u_{i-1}, u_i), because H_0 is connected and the weight of this super-edge is less than the weight of any external-edges. Thus Q is a path in H_0. Now, assume that there exists a path Q' in H_0, from u_0 to u_t, that is shorter than Q. Then, from Q' we can construct a shorter path in G (shorter than P) from the last vertex of U_0 to the first vertex of U_t. This is due to the fact that each super-node is connected and internal-edges have weight 0. This contradicts that P is a shortest path, hence Q is a shortest path in H_0.

It remains to show that S_H is a half-separator of H. Observe that for $i \neq 0$, $\omega_H(H_i) \leqslant \omega_H(H)/2$ because $\omega(H_0)$ is maximum. Let X_{H_0} be the set of vertices of any component in $H_0 \setminus S_H$. Then, there must exists a component X_G in $G \setminus S_G$ wholly containing all the super-nodes of the vertices of X_{H_0}. Let v be a vertex of X_{H_0} whose its super-node belongs to none component of $G \setminus S_G$. Then, there exists a vertex of this super-node that is in S_G. From our construction, v belongs to S_H (vertices of Q and super-nodes of P correspond): contradiction. Therefore, $\omega_H(X_{H_0}) \leqslant \omega_G(X_G)$. Moreover, $\omega_G(X_G) \leqslant \omega_G(G)/2 = \omega_H(H)/2$. Thus, S_H is a half-separator of H, completing the proof. □

4.2 One-Path Separable Graphs

In this part, we concentrate our attention to the graphs that belong to PS_1. We have seen in Proposition 2, that the subdivisions of outerplanar graph or of K_4 are face-separable, and thus belong to PS_2 (and even to SPS_2). Actually, outerplanar graphs are in PS_1:

Proposition 6. *Every weighted outerplanar graph is* 1*-path separable.*

Unfortunately, Proposition 6 does not generalize to treewidth-2 graphs. As depicted on Fig. 3, there are simple series-parallel graphs and subdivisions of K_4 that are not in PS_1.

The family PS_1 does not reduce to outerplanar graphs, as shown in Proposition 7. A *globe graph* is a subdivision of $K_{2,r}$, for some r, in which the two degree-r vertices may be adjacent.

Proposition 7. *Every weighted globe graph is* 1*-path separable.*

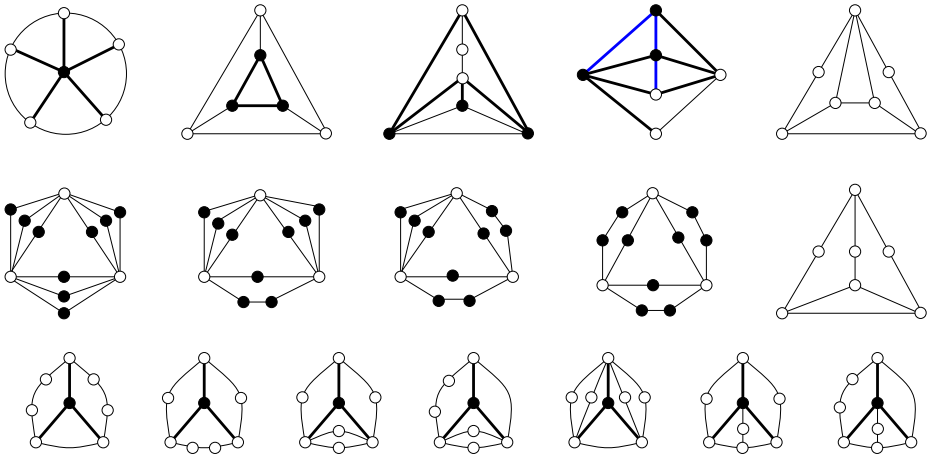


Fig. 3. Forbidden minors for planar graphs in PS_1. Black vertices and bold edges have weight 2, blue edges have weight 3, other vertices and edges have weight 1. The two non-planar forbidden minors are K_5, and a $K_{3,3}$ whose one edge is subdivided into two edges.

A first attempt to characterize PS_1 is given by Theorem 3.

Theorem 3. *Every biconnected graph of* PS_1 *is either isomorphic to* $K_{3,3}$*, or planar and excludes the list of minors depicted on Fig. 3.*

Proof. Let (G, ω) be a biconnected weighted graph with $G \in \mathrm{PS}_1$. First assume that G is not planar. From Kuratowski's criteria, G contains a subdivision of K_5 or $K_{3,3}$.

The complete graph K_5 is not 1-path separable graph from Proposition 1 (cf. Point 4 with $k = 2$). From Theorem 2, it follows that G cannot contain a subdivision of K_5, so it must contain a subdivision of $K_{3,3}$.

We shall proof that a subdivision of $K_{3,3}$ in which only one edge is subdivided into two edges is not 1-path separable. Denote by M this graph, and set unitary all the weights so that the total vertex-weight is 7. The diameter of M is two. The deletion of any shortest path deletes at most three vertices. Moreover, such a deletion cannot disconnect M, and thus leaves a component with at least $4 > 7/2$ vertices. M is not 1-path separable. Therefore, if G is not planar, then G can only be isomorphic to $K_{3,3}$.

We prove now that $K_{3,3}$ is 1-path separable. Denote by $\{x_1, x_2, x_3\}$ and $\{y_1, y_2, y_3\}$ the vertex set of each part of $K_{3,3}$. We first show that $K_{3,3}$ has a 1-path separator. Later we will prove it for all its induced subgraphs. W.l.o.g. assume that $\omega(x_1) \geqslant \omega(x_2) \geqslant \omega(x_3)$.

Define P_1 be a shortest path from x_1 to x_2, and assume P_1 contains y_{i_1}. Define P_2 be a shortest path from y_{i_2} to y_{i_3} (with i_1, i_2, i_3 pairwise different indices), and assume it contains x_{j_1} (denote by j_2, j_3 the two other x's indices). We show that P_1 or P_2 is a 1-path separator. By contradiction, if P_1 is not a half-separator,

then (and similarly for P_2) $\omega(P_1) < \omega(K_{3,3})/2$ and $\omega(G \setminus P_1) > \omega(K_{3,3})/2$. As $\omega(P_1)$ is lower bounded by $\omega(x_1)+\omega(x_2)+\omega(y_{i_1})$ and $\omega(G \setminus P_1)$ upper bounded by $\omega(x_3)+\omega(y_{i_2})+\omega(y_{i_3})$ (and similarly for P_2), it follows that:

$$\omega(x_1)+\omega(x_2)+\omega(y_{i_1}) < \omega(x_3)+\omega(y_{i_2})+\omega(y_{i_3}) \quad (1)$$

$$\omega(x_{j_1})+\omega(y_{i_2})+\omega(y_{i_3}) < \omega(x_{j_2})+\omega(x_{j_3})+\omega(y_{i_1}) \quad (2)$$

By summing these equations, we obtain:

$$\begin{aligned} &\omega(x_1)+\omega(x_2)+\omega(x_{j_1}) < \omega(x_3)+\omega(x_{j_2})+\omega(x_{j_3}) \\ \Rightarrow \quad &\omega(x_1)+\omega(x_2)+\omega(x_3) < \omega(x_3)+\omega(x_{j_2})+\omega(x_{j_3}) \leqslant \omega(x_3)+\omega(x_2)+\omega(x_1) \end{aligned}$$

since, by assumption, $\omega(x_3) \leqslant \omega(x_{j_1})$ and $\omega(x_{j_2})+\omega(x_{j_3}) \leqslant \omega(x_2)+\omega(x_1)$. This leads to a contradiction. Thus, P_1 or P_2 is a 1-path separator for $K_{3,3}$.

Now, let H be any induced subgraph of $K_{3,3}$. We use similar notations excepted that the two vertex-sets are $\{x_1,\ldots,x_p\}$ and $\{y_1,\ldots,y_q\}$ with $1 \leqslant p \leqslant q \leqslant 3$. If $p+q \leqslant 4$, then H is outerplanar, and thus 1-path separable by Proposition 6. We are left with the case $p=2$ and $q=3$, the case $p=q=3$ is already done. We define similarly the two paths P_1 and P_2, i.e., a shortest path from x_1 to x_2 containing y_{i_1}, and the shortest path between y's vertices different from y_{i_1} and through x_{j_1}. If these both paths are not half-separators, then Eq. (1) and (2) rewrite in (vertices x_3 and x_{j_3} do not exist anymore):

$$\omega(x_1)+\omega(x_2)+\omega(y_{i_1}) < \omega(y_{i_2})+\omega(y_{i_3})$$

$$\omega(x_{j_1})+\omega(y_{i_2})+\omega(y_{i_3}) < \omega(x_{j_2})+\omega(y_{i_1})$$

Summing these equations, we obtain:

$$\omega(x_1)+\omega(x_2)+\omega(x_{j_1}) < \omega(x_{j_2}),$$

a contradiction since vertex-weights are non-negative and $\omega(x_{j_2}) \leqslant \omega(x_1)$. Thus, P_1 or P_2 is a 1-path separator for H.

Therefore, we have proved that the only non-planar graph of PS_1 is $K_{3,3}$. For planar graphs, we manage to find forbidden minors represented in Fig. 3. To prove that each minor M of this list is indeed excluded, we exhibit a particular weight function ω for M. Actually, each vertex and edge has weight 1 or 2 as depicted on Fig. 3. We then exhaustively check that, for each pair u, v of vertices of M, the deletion of any shortest path from u to v leaves a component of weight $> \omega(M)/2$.

To illustrate this, consider for instance the "wheel graph", composed of a cycle of length 5 and a degree-5 vertex, called hereafter center, connected of all vertices of the cycle. The total weight of the graph is 7, the center has weight 2. Any shortest path from the center to a non-center vertex consists of one edge. Therefore its deletion leaves a path of 4 vertices, so of weight 4. Any shortest path between two non-center vertices consists of 2 edges at most, so leaving a component with the center and two (or more) non-center vertices, thus of weight at least 4. In both cases, the weight is $> 7/2$. This graph has no half-separator composed of a shortest path, and thus is not in PS_1. □

5 Conclusion

In this paper we have investigated the family of graphs that are k-path separable. Graph Minor Theory implies that such a family can be characterized by a finite set of forbidden minors that we have started to list for $k = 1$.

We propose here a list of further researches.

1. Determine the full list of forbidden minors for k-path separable graphs and for $k = 1$.
2. Find an explicit linear time algorithm to determine if a graph is k-path separable, for fixed k.
3. Prove or disprove that planar graphs are 2-path separable.
4. Prove NP-completeness for the problem of determining whether a given *weighted* graph has a k-path separator.
5. Extend the study to more general isometric separators, not only shortest paths.

References

1. Abraham, I., Gavoille, C.: Object location using path separators. In: 25^{th} Annual ACM Symp. on Principles of Distributed Comp. (PODC), pp. 188–197 (2006)
2. Abu-Khzam, F.N., Collins, R.L., Fellows, M.R., Langston, M.A., Suters, W.H., Symons, C.T.: Kernelization algorithms for the vertex cover problem: Theory and experiments. In: 6^{th} Workshop ALENEX, pp. 62–69 (2004)
3. Chepoi, V.D., Dragan, F.F., Estellon, B., Habib, M., Vaxès, Y.: Diameters, centers, and approximating trees of delta-hyperbolic geodesic spaces and graphs. In: 24^{st} Annual ACM Symposium on Computational Geometry (SoCG), pp. 59–68 (2008)
4. Dourisboure, Y., Gavoille, C.: Tree-decompositions with bags of small diameter. Discrete Mathematics 307(16), 2008–2029 (2007)
5. Flum, J., Grohe, M.: Parametrized Complexity Theory. Springer, Heidelberg (2006)
6. Koster, A.M.C.A., Bodlaender, H.L., van Hoesel, S.P.M.: Treewidth: Computational experiments. In: Elsevier (ed.) 1^{st} Cologne-Twente Workshop on Graphs and Combinatorial Optimization, vol. 8, pp. 54–57. ENDM, Amsterdam (2001)
7. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics 36(2), 177–189 (1979)
8. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. Journal of Combinatorial Theory, Series B 63(1), 65–110 (1995)
9. Robertson, N., Seymour, P.D.: Graph minors. XVI. Excluding a non-planar graph. Journal of Combinatorial Theory, Series B 89(1), 43–76 (2003)
10. Robertson, N., Seymour, P.D.: Graph minors. XX. Wagner's conjecture. Journal of Combinatorial Theory, Series B 92(2), 325–357 (2004)
11. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. Journal of the ACM 51(6), 993–1024 (2004)
12. Umezawa, K., Yamazaki, K.: Tree-length equals branch-length. Discrete Mathematics 309(13), 4656–4660 (2009)

Minimum Cost Edge-Colorings of Trees Can Be Reduced to Matchings

Takehiro Ito[1], Naoki Sakamoto[1], Xiao Zhou[1], and Takao Nishizeki[2]

[1] Graduate School of Information Sciences, Tohoku University,
Aoba-yama 6-6-05, Sendai, 980-8579, Japan
takehiro@ecei.tohoku.ac.jp,
sakamoto@nishizeki.ecei.tohoku.ac.jp,
zhou@ecei.tohoku.ac.jp
[2] School of Science and Technology, Kwansei Gakuin University,
2-1 Gakuen, Sanda, 669-1337, Japan
nishi@kwansei.ac.jp

Abstract. Let C be a set of colors, and let $\omega(c)$ be an integer cost assigned to a color c in C. An edge-coloring of a graph G is to color all the edges of G so that any two adjacent edges are colored with different colors in C. The cost $\omega(f)$ of an edge-coloring f of G is the sum of costs $\omega(f(e))$ of colors $f(e)$ assigned to all edges e in G. An edge-coloring f of G is optimal if $\omega(f)$ is minimum among all edge-colorings of G. In this paper, we show that the problem of finding an optimal edge-coloring of a tree T can be simply reduced in polynomial time to the minimum weight perfect matching problem for a new bipartite graph constructed from T. The reduction immediately yields an efficient simple algorithm to find an optimal edge-coloring of T in time $O(n^{1.5}\Delta \log(nN_\omega))$, where n is the number of vertices in T, Δ is the maximum degree of T, and N_ω is the maximum absolute cost $|\omega(c)|$ of colors c in C. We then show that our result can be extended for multitrees.

1 Introduction

Let $G = (V, E)$ be a graph with vertex set V and edge set E, and let C be a set of colors. An *edge-coloring* of G is to color all the edges in E so that any two adjacent edges are colored with different colors in C. The minimum number of colors required for edge-colorings of G is called the *chromatic index*, and is denoted by $\chi'(G)$. It is well-known that $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$ for every simple graph G and that $\chi'(G) = \Delta(G)$ for every bipartite (multi)graph G, where $\Delta(G)$ is the maximum degree of G [9]. The ordinary *edge-coloring problem* is to compute the chromatic index $\chi'(G)$ of a given graph G and find an edge-coloring of G using $\chi'(G)$ colors. Let ω be a cost function which assigns an integer $\omega(c)$ to each color $c \in C$, then the *cost edge-coloring problem* is to find an *optimal edge-coloring* of G, that is, an edge-coloring f such that $\sum_{e \in E} \omega(f(e))$ is minimum among all edge-colorings of G. An optimal edge-coloring does not always use the minimum number $\chi'(G)$ of colors. For example, suppose that $\omega(c_1) = 1$ and

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 274–284, 2010.

Fig. 1. (a) Edge-coloring using $\chi'(G) = 3$ colors, and (b) optimal edge-coloring using $\chi'(G) + 1 = 4$ colors, where $\omega(c_1) = 1$ and $\omega(c_2) = \omega(c_3) = \omega(c_4) = 5$

$\omega(c_i) = 5$ for each index $i \geq 2$, then the graph G with $\chi'(G) = 3$ in Fig. 1(a) can be uniquely colored with the three cheapest colors c_1, c_2 and c_3 as in Fig. 1(a), but this edge-coloring is not optimal; an optimal edge-coloring of G uses the four cheapest colors c_1, c_2, c_3 and c_4, as illustrated in Fig. 1(b). However, every simple graph G has an optimal edge-coloring using $\Delta(G)$ or $\Delta(G)+1$ colors [5,8], and every bipartite (multi)graph G and hence every tree has an optimal edge-coloring using $\Delta(G)$ $(= \chi'(G))$ colors [1,5]. The edge-chromatic sum problem, introduced by Giaro and Kubale [4], is merely the cost edge-coloring problem for the special case where $\omega(c_i) = i$ for each integer $i \geq 1$.

The cost edge-coloring problem has a natural application for scheduling [10]. Consider the scheduling of biprocessor tasks of unit execution time on dedicated machines. An example of such tasks is the file transfer problem in a computer network in which each file engages two corresponding nodes, sender and receiver, simultaneously [2]. Another example is the biprocessor diagnostic problem in which links execute concurrently the same test for a fault tolerant multiprocessor system [6]. These problems can be modeled by a graph G in which machines correspond to the vertices and tasks correspond to the edges. An edge-coloring of G corresponds to a schedule, where the edges colored with color $c_i \in C$ represent the collection of tasks that are executed in the ith time slot. Suppose that a task executed in the ith time slot takes the cost $\omega(c_i)$. Then the goal is to find a schedule that minimizes the total cost, and hence this corresponds to the cost edge-coloring problem.

The cost edge-coloring problem is APX-hard even for bipartite graphs [7], and hence there is no polynomial-time approximation scheme (PTAS) for the problem unless P = NP. On the other hand, Zhou and Nishizeki gave an algorithm to solve the cost edge-coloring problem for trees T in time $O(n\Delta^{1.5} \log(nN_\omega))$, where n is the number of vertices in T, Δ is the maximum degree of T, and N_ω is the maximum absolute cost $|\omega(c)|$ of colors c in C [10]. The algorithm is based on a dynamic programming approach, and computes a DP table for each vertex of a given tree T from the leaves to the root of T. For computing the DP tables, the algorithm needs to construct $O(n)$ bipartite graphs in total and solves the minimum weight perfect matching problem for each of them.

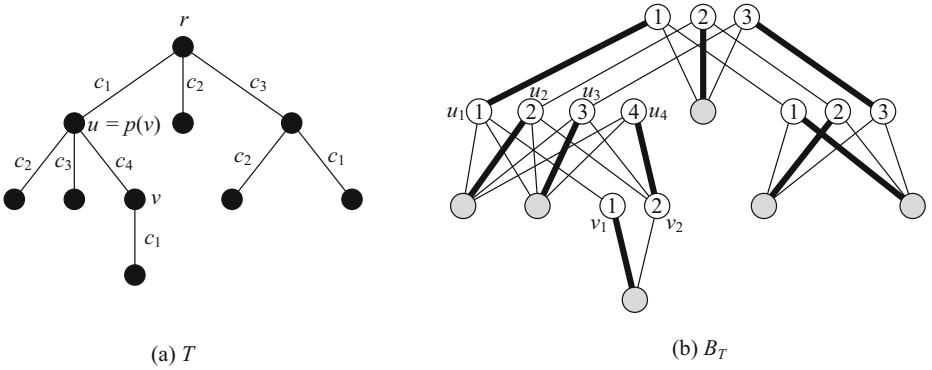


Fig. 2. (a) Optimal compact edge-coloring of a tree T, and (b) perfect matching of B_T, whose edges are drawn by thick lines

In this paper, we first show that the cost edge-coloring problem for a tree T can be simply reduced in polynomial time to the problem of finding a minimum weight perfect matching in an edge-weighted bipartite graph B_T constructed from T, as illustrated in Fig. 2. The reduction takes time $O(n\Delta)$, and yields an efficient simple algorithm to find an optimal edge-coloring of T in time $O(n^{1.5}\Delta \log(nN_\omega))$. Our algorithm constructs a single bipartite graph B_T, and solves only once the minimum weight perfect matching problem for B_T. Thus, our algorithm is much simpler than the known algorithm [10], and can be easily implemented. We then show that the algorithm for trees can be extended for multitrees, which will be defined in Section 5.

The rest of the paper is organized as follows. In Section 2 we first define some basic terms which will be used throughout the paper. We then give the reduction in Section 3. In Section 4 we prove a lemma used by the reduction. In Section 5 we show that the algorithm for trees can be extended for multitrees. Finally, in Section 6 we give a conclusion.

2 Preliminaries

In this section, we define some basic terms.

Let $T = (V, E)$ be a tree with a set V of vertices and a set E of edges. We sometimes denote by $V(T)$ and $E(T)$ the vertex set and the edge set of T, respectively. We choose an arbitrary vertex r of T as the *root*, and regard T as a rooted tree. We denote by n the number of vertices in T, that is, $n = |V|$. One may assume that $n \geq 2$. The *degree* $d(v)$ of a vertex v is the number of edges in E incident to v. We denote the maximum degree of T by $\Delta(T)$ or simply by Δ. We denote by $\mathrm{ch}(v)$ the number of edges joining a vertex v and its children in T. Then, $\mathrm{ch}(r) = d(r)$, and $\mathrm{ch}(v) = d(v) - 1$ for every vertex $v \in V \setminus \{r\}$. We denote by $p(v)$ the parent of a vertex $v \in V \setminus \{r\}$ in T.

Although T has an optimal edge-coloring using $\Delta(T)$ colors [1,5], we assume for the sake of convenience that $|C| = \Delta(T) + 1$, and we write $C = \{c_1, c_2, \cdots, c_{\Delta+1}\}$. An *edge-coloring* $f : E \to C$ of a tree $T = (V, E)$ is to color all the edges of T by colors in C so that any two adjacent edges are colored with different colors. Let $\omega : C \to \mathbb{Z}$ be a *cost function*, where $\mathbb{Z}$ is the set of all integers. One may assume without loss of generality that ω is non-decreasing, that is, $\omega(c_i) \leq \omega(c_{i+1})$ for every index i, $1 \leq i \leq \Delta$. The *cost* $\omega(f)$ *of an edge-coloring* f of a tree $T = (V, E)$ is defined as follows:

$$\omega(f) = \sum_{e \in E} \omega(f(e)).$$

An edge-coloring f of T is *optimal* if $\omega(f)$ is minimum among all edge-colorings of T. The *cost edge-coloring problem* is to find an optimal edge-coloring of a given tree.

For an edge-coloring f of a tree T and a vertex v of T, we denote by $C(f, v)$ the set of all colors that are assigned to the edges incident to v, that is,

$$C(f, v) = \{f(e) \mid e \text{ is an edge incident to } v \text{ in } T\}.$$

We say that a color $c \in C$ is *missing at* v if $c \notin C(f, v)$. We denote by $\mathrm{Miss}(f, v)$ the set of all colors missing at v, that is, $\mathrm{Miss}(f, v) = C \setminus C(f, v)$.

Interchanging colors in an "alternating path" is one of the standard techniques for ordinary edge-colorings [9], which we also use in the paper. Let f be an edge-coloring of a tree T, let c_α and c_β be any two colors in C, and let $T(c_\alpha, c_\beta)$ be the subgraph of T induced by all edges colored with c_α or c_β. Since T is a tree, each connected component of $T(c_\alpha, c_\beta)$ is a path, called a $c_\alpha c_\beta$-*alternating path*, whose edges are colored alternately with c_α and c_β. A vertex $v \in V$ is an end of a $c_\alpha c_\beta$-alternating path if and only if exactly one of the two colors c_α and c_β is missing at v. We denote by $P(v; c_\alpha, c_\beta)$ a $c_\alpha c_\beta$-alternating path starting with v. Interchanging colors c_α and c_β in $P(v; c_\alpha, c_\beta)$, one can obtain another edge-coloring f' of T.

For a graph $G = (V, E)$, a subset M of E is called a *matching* of G if no two edges in M share a common vertex. A matching M of G is *perfect* if every vertex of G is an end of an edge in M. Thus, $|M| = \frac{1}{2}|V|$ for every perfect matching M of G. Let $w : E \to \mathbb{Z}$ be a weight function which assigns an integer weight $w(e) \in \mathbb{Z}$ to each edge e in G. Then, the *weight* $w(M)$ *of a matching* M of G is defined as follows:

$$w(M) = \sum_{e \in M} w(e).$$

The *minimum weight perfect matching problem* is to find a perfect matching M of a given graph G such that $w(M)$ is minimum among all perfect matchings in G. The problem can be solved for a bipartite graph $G = (V, E)$ in time $O(\sqrt{|V|}|E| \log(|V|N_w))$, where N_w is the maximum absolute weight $|w(e)|$ of edges e in E [3].

3 Reduction

Our main result is the following.

Theorem 1. *The cost edge-coloring problem for a tree T can be reduced in time $O(n\Delta)$ to the minimum weight perfect matching problem for a single bipartite graph B_T constructed from T.*

Before presenting the reduction, we introduce a "compact" edge-coloring of a tree. Let $T = (V, E)$ be a tree with root r. An edge-coloring f of T is *compact* if the following two conditions (i) and (ii) hold:

(i) for the root r of T, $C(f, r) = \{c_1, c_2, \cdots, c_{\mathrm{ch}(r)}\}$; and
(ii) for each vertex $v \in V \setminus \{r\}$, $C(f, v) = \{c_1, c_2, \cdots, c_{\mathrm{ch}(v)}, c_k\}$ for some index k such that
  (a) $k \geq \mathrm{ch}(v) + 1$; and
  (b) if $k \geq d(v) + 1$, then $k \leq d(u)$ and c_k is assigned to the edge joining v and the parent $u = p(v)$.

For example, the edge-coloring in Fig. 2(a) is compact. Clearly, a compact edge-coloring uses colors $c_1, c_2, \cdots, c_\Delta$ and does not use color $c_{\Delta+1}$. We then have the following lemma, whose proof will be given in Section 4.

Lemma 1. *Every tree T has an optimal edge-coloring which is compact.*

We now give the reduction from the cost edge-coloring problem for a tree T to the minimum weight perfect matching problem for a bipartite graph B_T.

The bipartite graph $B_T = (V_B, E_B)$ can be constructed from a tree $T = (V, E)$, as follows. (See Figs. 2 and 3.)

(i) For each vertex $v \in V$, add $d(v)$ vertices $v_1, v_2, \cdots, v_{d(v)}$ to V_B.
(ii) For each edge $(u, v) \in E$ with $u = p(v)$, add $d(u)$ edges to E_B, as follows: for each index i, $1 \leq i \leq d(u)$, join vertices u_i and v_j by an edge whose weight is $w((u_i, v_j)) = \omega(c_i)$, where

$$j = \begin{cases} i & \text{if } i \leq d(v); \\ d(v) & \text{otherwise.} \end{cases}$$

Clearly, $|V_B| = \sum_{v \in V} d(v) = 2(n-1)$ and $|E_B| = \sum_{(u,v) \in E} d(u) = O(n\Delta)$. Therefore, the bipartite graph B_T can be constructed from T in time $O(n\Delta)$.

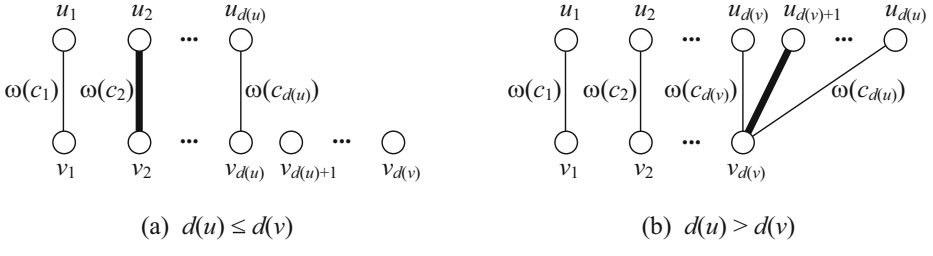


Fig. 3. Subgraph $B_T(u, v)$ of B_T corresponding to an edge (u, v) of T

Clearly, the maximum absolute weight $N_w = \max\{|\omega(c_1)|, |\omega(c_\Delta)|\}$ of edges in B_T is not greater than the maximum absolute cost $N_\omega = \max\{|\omega(c_1)|, |\omega(c_{\Delta+1})|\}$ of colors in C.

For each edge (u, v) in T, we denote by $B_T(u, v)$ the subgraph of B_T induced by vertices $u_1, u_2, \cdots, u_{d(u)}$ and $v_1, v_2, \cdots, v_{d(v)}$. $B_T(u, v)$ corresponds to edge (u, v) of T. (See Fig. 3.) We then have the following lemma.

Lemma 2. *For every tree T, the following* (a) *and* (b) *hold*:

(a) *every perfect matching M of B_T contains exactly one of the edges in $B_T(u, v)$ for every edge (u, v) of T, as illustrated in Fig. 3 where edges in M are drawn by thick lines*; *and*

(b) *every perfect matching M of B_T induces a compact edge-coloring f of T. Conversely, every compact edge-coloring f of T induces a perfect matching M of B_T. Furthermore, $\omega(f) = w(M)$.*

Proof. (a) Let M be a perfect matching of B_T. We prove from the leaves to the root that M contains exactly one of the edges of $B_T(u, v)$. One may assume that $u = p(v)$.

If v is a leaf of T, then $B_T(u, v)$ is a star with center v_1 and only the edges of $B_T(u, v)$ are incident to v_1 in B_T. Therefore, the perfect matching M of B_T contains exactly one edge of $B_T(u, v)$, say (u_k, v_1) for some index k, $1 \le k \le d(u)$.

One may thus assume that v is an internal vertex of T, and that M contains exactly one of the edges of $B_T(v, w)$ for each child w of v in T. Since v has a parent u in T, we have $v \neq r$ and hence $\mathrm{ch}(v) = d(v) - 1$. Therefore, M contains exactly $d(v) - 1$ edges in the bipartite subgraphs corresponding to the edges of T joining v and its $d(v) - 1$ children. Hence, exactly one of the vertices $v_1, v_2, \cdots, v_{d(v)}$, say v_j, is not an end of these $d(v) - 1$ edges in M. Since M is a perfect matching of B_T, M contains exactly one edge (u_k, v_j) of $B_T(u, v)$ for some index k, $1 \le k \le d(u)$.

(b) Every perfect matching M of B_T induces an edge-coloring f of T, in which each edge (u, v) of T is colored with c_k for the index k above; the edge of $B_T(u, v)$ contained in M has an end u_k, $1 \le k \le d(u)$. One can easily observe that the edge-coloring f is compact.

Conversely, every compact edge-coloring f of T induces a perfect matching M of B_T; if $u = p(v)$ and $f((u, v)) = c_i$, $1 \le i \le d(u)$, then M contains an edge joining u_i and v_j where

$$j = \begin{cases} i & \text{if } i \le d(v); \\ d(v) & \text{otherwise.} \end{cases}$$

Obviously, $\omega(f) = w(M)$. □

By Lemma 1 every tree T has an optimal edge-coloring f which is compact, and hence by Lemma 2(b) B_T has a perfect matching M such that $w(M) = \omega(f)$. Remember that $|V_B| = O(n)$, $|E_B| = O(n\Delta)$, and the maximum absolute weight N_w of edges in B_T is not greater than the maximum absolute cost N_ω of colors in C. Since a minimum weight perfect matching of B_T can be found in time $O(\sqrt{|V_B|}|E_B| \log(|V_B|N_w))$ [3], we can find an optimal edge-coloring of T in time $O(n^{1.5}\Delta \log(nN_\omega))$.

4 Proof of Lemma 1

In this section, we give a proof of Lemma 1.

Let $T = (V, E)$ be a tree with root r. For a vertex w of T, we denote by T_w the subtree of T which is rooted at w and is induced by w and all descendants of w in T. (See Fig. 4(a).) Clearly, $T = T_r$.

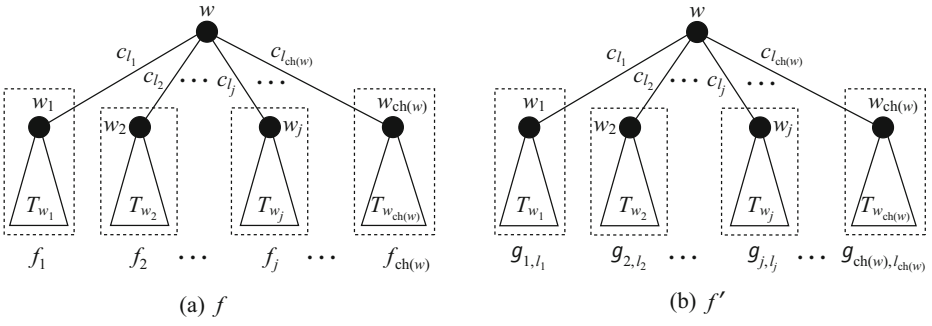


Fig. 4. (a) A (w, i)-compact edge-coloring f of T_w, and (b) a (T_w, i)-compact edge-coloring f' of T_w

Let w be an arbitrary vertex of T. Since $\chi'(T_w) \leq \Delta(T)$ and $|C| = \Delta(T) + 1$, for each color $c_i \in C$, T_w has an edge-coloring f in which c_i is not used and hence $c_i \in \mathrm{Miss}(f, w)$. Let

$$\omega(T_w, i) = \min\{\omega(f) \mid f \text{ is an edge-coloring of } T_w \text{ such that } c_i \in \mathrm{Miss}(f, w)\}.$$

For a color $c_i \in C$, an edge-coloring f of T_w is defined to be (w, i)-*compact* if the following two conditions (i) and (ii) hold:

(i) $c_i \in \mathrm{Miss}(f, w)$; and

(ii) if $i \geq \mathrm{ch}(w) + 1$ then $C(f, w) = \{c_1, c_2, \cdots, c_{\mathrm{ch}(w)}\}$, and otherwise $C(f, w) \cup \{c_i\} = \{c_1, c_2, \cdots, c_{\mathrm{ch}(w)+1}\}$.

We then have the following lemma.

Lemma 3. *For each color $c_i \in C$, T_w has a (w, i)-compact edge-coloring f such that $\omega(f) = \omega(T_w, i)$.*

Proof. We give a proof only for the case where $i \geq \mathrm{ch}(w) + 1$. (The proof for the other case is similar.) The definition of $\omega(T_w, i)$ implies that T_w has an edge-coloring f such that $c_i \in \mathrm{Miss}(f, w)$ and $\omega(f) = \omega(T_w, i)$. In particular, let f be an edge-coloring of T_w such that $|C(f, w) \cap \{c_1, c_2, \cdots, c_{\mathrm{ch}(w)}\}|$ is maximum among all these edge-colorings. Suppose for a contradiction that f is not (w, i)-compact. Then, $C(f, w) \neq \{c_1, c_2, \cdots, c_{\mathrm{ch}(w)}\}$. Since $|C(f, w)| = \mathrm{ch}(w)$, there exist two colors c_α and c_β such that

$$c_\alpha \in \{c_1, c_2, \cdots, c_{\mathrm{ch}(w)}\} \setminus C(f, w)$$

and

$$c_\beta \in C(f,w) \setminus \{c_1, c_2, \cdots, c_{\mathrm{ch}(w)}\}.$$

Since $\alpha \leq \mathrm{ch}(w) < \beta$, we have $\omega(c_\alpha) \leq \omega(c_\beta)$. Since $i \geq \mathrm{ch}(w)+1$, we have $c_\alpha \neq c_i$. Since $c_i \in \mathrm{Miss}(f,w)$ and $c_\beta \in C(f,w)$, we have $c_\beta \neq c_i$. Since $c_\alpha \in \mathrm{Miss}(f,w)$ and $c_\beta \in C(f,w)$, there is a $c_\alpha c_\beta$-alternating path $P(w; c_\alpha, c_\beta)$ starting from w. We obtain another edge-coloring f' of T_w by interchanging colors c_α and c_β in $P(w; c_\alpha, c_\beta)$. Since $\omega(c_\alpha) \leq \omega(c_\beta)$, $\omega(f') \leq \omega(f)$. Since $c_i \neq c_\alpha, c_\beta$ and $c_i \in \mathrm{Miss}(f,w)$, we have $c_i \in \mathrm{Miss}(f',w)$ and hence $\omega(T_w,i) \leq \omega(f')$. Therefore, $\omega(T_w,i) \leq \omega(f') \leq \omega(f) = \omega(T_w,i)$ and hence $\omega(f') = \omega(T_w,i)$. Since $c_\alpha \in C(f',w)$ and $\alpha \leq \mathrm{ch}(w) < \beta$, we have

$$C(f',w) \cap \{c_1, c_2, \cdots, c_{\mathrm{ch}(w)}\} = \Big(C(f,w) \cap \{c_1, c_2, \cdots, c_{\mathrm{ch}(w)}\}\Big) \cup \{c_\alpha\}$$

and hence

$$|C(f',w) \cap \{c_1, c_2, \cdots, c_{\mathrm{ch}(w)}\}| > |C(f,w) \cap \{c_1, c_2, \cdots, c_{\mathrm{ch}(w)}\}|,$$

a contradiction. □

A (w,i)-compact edge-coloring f of T_w is defined to be (T_w,i)-*compact* if the following condition (iii) holds:

(iii) for each vertex $v \in V(T_w) \setminus \{w\}$, $C(f,v) = \{c_1, c_2, \cdots, c_{\mathrm{ch}(v)}, c_k\}$ for some index k such that
  (a) $k \geq \mathrm{ch}(v)+1$; and
  (b) if $k \geq d(v)+1$, then $k \leq d(u)$ and c_k is assigned to the edge joining v and the parent $u = p(v)$.

Clearly, an edge-coloring f of T with root r is compact if f is $(T_r, \mathrm{ch}(r)+1)$-compact. Using the argument on an alternating path, one can easily show that the cost $\omega(f)$ of an optimal edge-coloring f of T is equal to $\omega(T_r, \mathrm{ch}(r)+1)$. Therefore, as a proof of Lemma 1, it suffices to prove the following lemma.

Lemma 4. *For each vertex w of T and each color $c_i \in C$, T_w has a (T_w,i)-compact edge-coloring f such that $\omega(f) = \omega(T_w,i)$.*

Proof. We prove the lemma by induction on the number of vertices in T_w.

For the base case, let w be a leaf of T. Then, T_w is a tree of a single vertex w, and hence the lemma trivially holds.

Let c_i be a color in C, and let w be an internal vertex of T. Let $w_1, w_2, \cdots, w_{\mathrm{ch}(w)}$ be the children of w, as illustrated in Fig. 4(a). Suppose as the induction hypothesis that the lemma holds for each color $c_l \in C$ and each subtree T_{w_j}, $1 \leq j \leq \mathrm{ch}(w)$. Then, for each color $c_l \in C$, T_{w_j} has a (T_{w_j}, l)-compact edge-coloring $g_{j,l}$ such that $\omega(g_{j,l}) = \omega(T_{w_j}, l)$.

By Lemma 3, T_w has a (w,i)-compact edge-coloring f such that $\omega(f) = \omega(T_w,i)$. If f is (T_w,i)-compact, then we have done. So we may assume that f is not (T_w,i)-compact. For each subtree T_{w_j}, $1 \leq j \leq \mathrm{ch}(w)$, let $f_j = f|T_{w_j}$ be the restriction of f to T_{w_j}, that is, $f_j(e) = f(e)$ for each edge e of T_{w_j}. Let c_{l_j} be the color assigned to the edge (w,w_j), $1 \leq j \leq \mathrm{ch}(w)$, by f, as illustrated in Fig. 4(a). Then one can easily observe that $c_{l_j} \in \mathrm{Miss}(f_j, w_j)$ and

$\omega(f_j) = \omega(T_{w_j}, l_j) = \omega(g_{j,l_i})$ for each j, $1 \leq j \leq \mathrm{ch}(w)$. We now construct another edge-coloring f' of T_w, as follows (see Fig. 4(b)):

$$f'(e) = \begin{cases} g_{j,l_j}(e) & \text{if } e \in E(T_{w_j}) \text{ for some } j,\ 1 \leq j \leq \mathrm{ch}(w); \\ f(e) & \text{otherwise.} \end{cases}$$

Clearly, f' is (T_w, i)-compact and $\omega(f') = \omega(f) = \omega(T_w, i)$. □

5 Multitrees

Replace each edge in a tree by multiple edges, as illustrated in Fig. 5(a). The resulting multigraph is called a *multitree*. In this section, we show that our reduction for trees can be extended for multitrees.

Theorem 2. *The cost edge-coloring problem for multitrees $T = (V, E)$ can be reduced in time $O(|E|\Delta)$ to the minimum weight perfect matching problem for a bipartite graph B_T, and can be solved in time $O(|E|^{1.5}\Delta \log(|E|N_\omega))$.*

Let $T = (V, E)$ be a multitree with root r. Since T is a bipartite multigraph, T has an optimal edge-coloring using Δ colors [1]. For a vertex $v \in V \setminus \{r\}$, we denote by $m(v)$ the number of multiple edges joining v and $p(v)$. Thus $m(v) = d(v) - \mathrm{ch}(v)$. Similarly as for trees, an edge-coloring f of a multitree T is defined to be *compact* if the following two conditions (i) and (ii) hold:

(i) for the root r of T, $C(f, r) = \{c_1, c_2, \cdots, c_{\mathrm{ch}(r)}\}$; and
(ii) for each vertex $v \in V \setminus \{r\}$, $C(f, v) = \{c_1, c_2, \cdots, c_{\mathrm{ch}(v)}, c_{k_1}, c_{k_2}, \cdots, c_{k_{m(v)}}\}$ for some indices k_j, $1 \leq j \leq m(v)$, such that
  (a) $k_j \geq \mathrm{ch}(v) + 1$; and
  (b) if $k_j \geq d(v) + 1$, then $k_j \leq d(u)$ and c_{k_j} is assigned to an edge joining v and the parent $u = p(v)$.

Figure 5(a) depicts a compact edge-coloring of a multitree. Clearly, a compact edge-coloring uses colors $c_1, c_2, \cdots, c_\Delta$ and does not use color $c_{\Delta+1}$. Similarly as Lemma 1, one can prove that every multitree has an optimal edge-coloring which is compact.

The bipartite graph $B_T = (V_B, E_B)$ for a multitree $T = (V, E)$ can be constructed as follows. (See Figs. 5 and 6.)

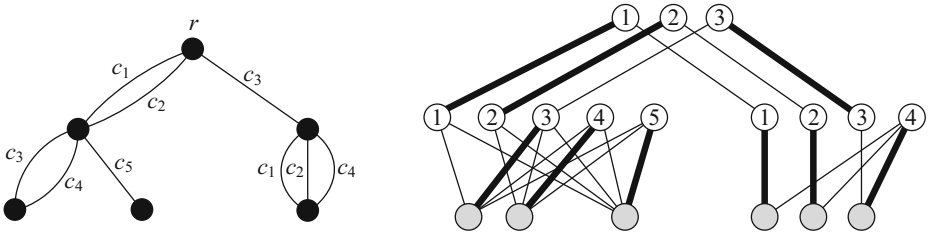


Fig. 5. (a) Optimal compact edge-coloring of a multitree T, and (b) its corresponding perfect matching in B_T whose edges are drawn by thick lines

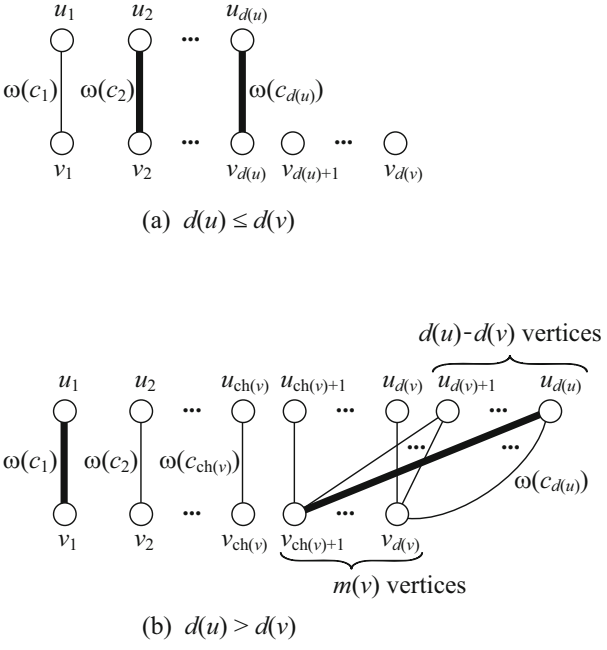


Fig. 6. Subgraph $B_T(u, v)$ of B_T corresponding to multiple edges joining v and $u = p(v)$ in T

(i) For each vertex $v \in V$, add $d(v)$ vertices $v_1, v_2, \cdots, v_{d(v)}$ to V_B.
(ii) For each set of $m(v)$ multiple edges joining vertices v and $u = p(v)$, add edges to E_B, as follows: for each index i, $1 \le i \le d(u)$, join vertices u_i and v_j by an edge whose weight is $w((u_i, v_j)) = \omega(c_i)$, where

$$j = \begin{cases} i & \text{if } i \le d(v); \\ \mathrm{ch}(v) + 1, \mathrm{ch}(v) + 2, \cdots, d(v) & \text{otherwise.} \end{cases}$$

Clearly, $|V_B| = 2|E|$. If $d(u) \le d(v)$, then $|E(B_T(u, v))| = d(u)$. If $d(u) > d(v)$, then $|E(B_T(u, v))| = d(v) + \big(d(u) - d(v)\big)m(v)$. In either case, $|E(B_T(u, v))| \le d(u)m(v)$ because $m(v) \ge 1$. Therefore, $|E_B| \le \sum d(u)m(v) = O(\Delta|E|)$, where the summention is taken over all pairs (u, v) such that $u = p(v)$.

Similarly as in Lemma 2, one can prove that every perfect matching M of B_T contains exactly $m(v)$ edges in $B_T(u, v)$; every compact edge-coloring f of a multitree T induces a perfect matching M of B_T, and *vice versa*; and $\omega(f) = w(M)$. Thus, our reduction for trees can be extended for multitrees, and hence Theorem 2 holds.

6 Conclusions

In this paper, we show that the cost edge-coloring problem for a tree T can be reduced in time $O(n\Delta)$ to the minimum weight perfect matching problem for

the bipartite graph B_T. This reduction immediately yields an algorithm which actually finds an optimal edge-coloring of T in time $O(n^{1.5}\Delta \log(nN_\omega))$. We then show that the algorithm for trees can be extended for multitrees.

References

1. Cardinal, J., Ravelomanana, V., Valencia-Pabon, M.: Chromatic edge strength of some multigraphs. Electronic Notes in Discrete Mathematics 30, 39–44 (2008)
2. Coffman, E.G., Garey, M.R., Johnson, D.S., LaPaugh, A.S.: Scheduling file transfers. SIAM J. Computing 14, 744–780 (1985)
3. Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for network problems. SIAM J. Computing 18, 1013–1036 (1989)
4. Giaro, K., Kubale, M.: Edge-chromatic sum of trees and bounded cyclicity graphs. Information Processing Letters 75, 65–69 (2000)
5. Hajiabolhassan, H., Mehrabadi, M.L., Tusserkani, R.: Minimal coloring and strength of graphs. Discrete Mathematics 215, 265–270 (2000)
6. Krawczyk, H., Kubale, M.: An approximation algorithm for diagnostic test scheduling in multicomputer systems. IEEE Trans. Computers 34, 869–872 (1985)
7. Marx, D.: Complexity results for minimum sum edge coloring. Discrete Applied Mathematics 157, 1034–1045 (2009)
8. Mitchem, J., Morriss, P., Schmeichel, E.: On the cost chromatic number of outerplanar, planar, and line graphs. Discussiones Mathematicae Graph Theory 17, 229–241 (1997)
9. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice Hall, New Jersey (2000)
10. Zhou, X., Nishizeki, T.: Algorithm for the cost edge-coloring of trees. J. Combinatorial Optimization 8, 97–108 (2004)

Computing Minimum Diameter Color-Spanning Sets

Rudolf Fleischer* and Xiaoming Xu**

School of Computer Science and IIPL, Fudan University, Shanghai, China
{rudolf,xuxiaoming}@fudan.edu.cn

Abstract. We study the minimum diameter color-spanning set problem which has recently drawn some attention in the database community. We show that the problem can be solved in polynomial time for L_1 and L_∞ metrics, while it is NP-hard for all other L_p metrics even in two dimensions. However, we can efficiently compute a constant factor approximation.

1 Introduction

Assume we have a set of resources of one of several different types, or colors. We want to solve a task that requires simultaneous use of one resource of each color. There is a communication delay between any pair of resources. How should we allocate the resources so as to minimize the maximum delay between any two of our selected resources? We call this the *minimum diameter color-spanning set problem (*MDCS*)*. It arises in large computer networks with different types of servers (think of a large company trying to pool resources to solve a certain computational task). It also arises in spatial databases, where it has recently been studied by Zhang et al. [7]. For example, we may search for a holiday location that features skiing, sailing, golfing, and shopping, all within short distance of each other and of our hotel.

Modeling the Problem. We model MDCS as follows. We are given a set S of n points in d-dimensional space $\mathbb{R}^d$. We measure distances in the L_p metric, for some $1 \leq p \leq \infty$. Each point is colored in one of k colors, where $k \geq 1$. S may be a multiset, which means we can have scenarios where a point is colored simultaneously with several colors. We call a subset of k points of distinct colors a *rainbow set*. MDCS is the problem of finding a rainbow set of smallest diameter. If we want to emphasize the dimension and metric, we write MDCS(d, p). We denote the smallest diameter of a rainbow set of S in L_p metric by $r_p(S)$. See Fig. 1 for an example.

* This work was supported by a grant from the NSFC (no. 60973026), the Shanghai Leading Academic Discipline Project (no. B114), and the Shanghai Committee of Science and Technology (nos. 08DZ2271800 and 09DZ2272800).

** The order of authors follows the international standard of alphabetic order of the family name; first-author order would be Xu and Fleischer.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 285–292, 2010.

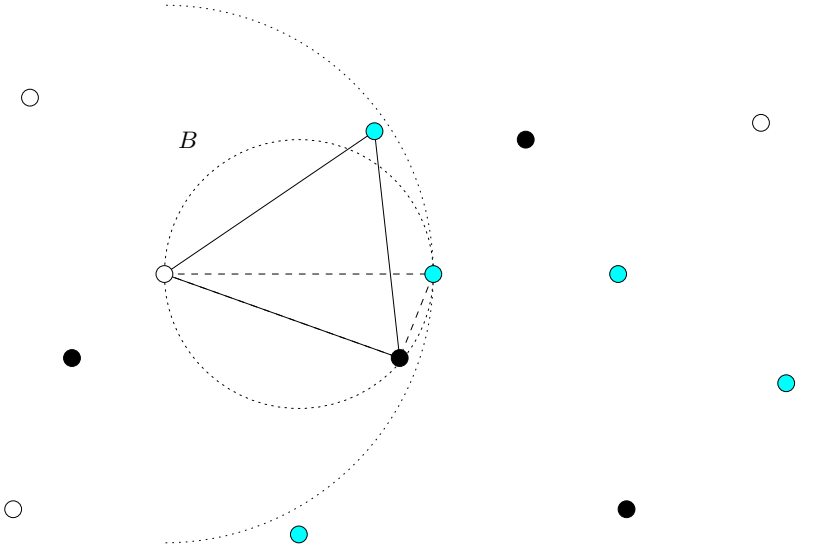


Fig. 1. An instance of MDCS(2, 2) with 3 colors. The smallest color-spanning disc B, enclosing the dashed triangle, does not minimize the diameter of a rainbow set, but it is a good approximation (Lemma 4). The solid triangle spans the minimum-diameter rainbow set.

Previous Work. Zhang et al. proposed an $O(n^k)$-time algorithm for MDCS$(2, 2)$ based on a brute-force enumeration of all possible rainbow sets. Their algorithm was implemented in a geographical tagging system named MarcoPolo by Chen et al. [3]. They also list several other applications of this problem.

Our Results. It is straightforward to solve the problem for $d = 1$ in $O(n \log n)$ time. On the other hand, we show that MDCS$(2, 2)$ is NP-hard for $d \geq 2$ and $1 < p < \infty$. We do not know whether the problem is $W[1]$-hard or APX-hard for these metrics, but there are efficient approximation algorithms. For example, we can approximate $r_2(S)$ by a factor of 1.154 in time $O(n \log n)$ in two dimensions. For $p \in \{1, \infty\}$, we can solve MDCS$(2, p)$ optimally in time $O(n \log n)$, and MDCS$(3, p)$ in time $O(k^{1+\epsilon} n^2)$, for any $\epsilon > 0$. In $\mathbb{R}^d$, the running time is $O(n^{d+2})$.

Related Work. To the best of our knowledge, this problem has not been studied from a theoretical point of view. Several other color-spanning set problems have been studied by Abellanas et al. They called rainbow sets *color-spanning sets*. They gave efficient algorithms in two dimensions for the smallest color-spanning disc problem [1] (finding a smallest disc containing at least one point of each color) and the smallest color-spanning rectangle problem [2] (finding a smallest rectangle containing at least one point of each color). We denote the diameter of a smallest color-spanning disc (or ball in higher dimensions) of a k-colored set S by $b_p(S)$.

In two dimensions, the smallest color-spanning disc for any L_p metric can be computed in time $O(kn \log n)$ [1]. This algorithm uses a farthest color Voronoi diagram which can be computed using an algorithm for the computation of the upper envelope of Voronoi surfaces in one dimension higher [5, Theorem 8]. In three dimensions, the running time becomes $O(k^{1+\epsilon}n^2)$, for any $\epsilon > 1$ [5, Theorem 19]. To generalize these algorithms to higher dimensions would require to bound the complexity of the envelopes of higher-dimensional Voronoi surfaces, which required considerable effort even in two and three dimensions. However, we can always find the smallest enclosing ball in time $O(n^{d+2})$ by brute-force enumeration of all balls with some subset of $d+1$ points on their boundary.

Structure of the Paper. In Section 2, we state a few facts about smallest enclosing balls in higher dimensions. In Section 3, we present efficient algorithms for $\texttt{MDCS}(d, 1)$ and $\texttt{MDCS}(d, \infty)$ and approximation algorithms for the other L_p metrics. In Section 4, we show that $\texttt{MDCS}(2, p)$ is NP-complete for $1 < p < \infty$. We conclude the paper in Section 5.

2 Preliminaries

We first state a few facts about smallest enclosing balls of point sets in higher dimensions. For a point set S, we denote its diameter in L_p metric by $diam_p(S)$ and its smallest enclosing ball by $B_p(S)$.

As we will see later, the worst case ratio of the diameter of the smallest enclosing ball of a point set and the diameter of the point set determines the approximation ratio of our algorithms. We denote this ratio by α_p^d in d-dimensional space with L_p metric. The following proposition is illustrated in Fig. 2.

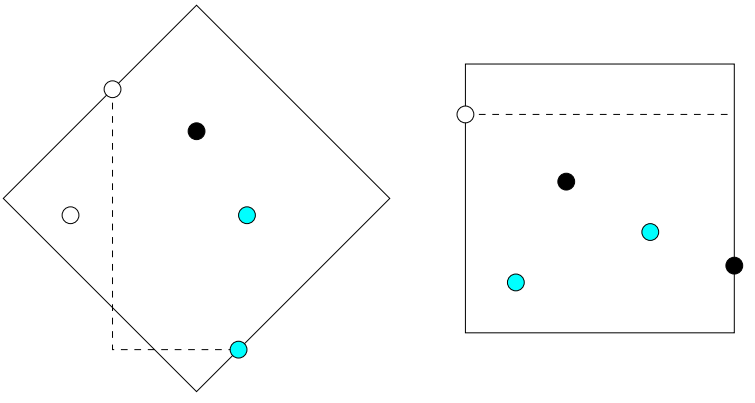


Fig. 2. The smallest spanning-color discs in L_1 metric (left) and L_∞ metric (right) for three-colored point sets. In both cases, the length of the dashed line (the distance of the two points on the boundary of the square) is equal to the diameter of the disc.

Proposition 1. *Let S be a point set in $\mathbb{R}^d$. For $p \in \{1, \infty\}$, $B_p(S)$ is a hypercube, and $diam_p(S) = diam_p(B_p(S))$, i.e., $\alpha_1^d = \alpha_\infty^d = 1$.* □

Proposition 2. *For $1 < p < \infty$, any two points on the boundary of a ball B in L_p metric have distance at most $diam_p(B)$, with equality if and only if the two points are antipodal.* □

Corollary 3. *Let S be a k-colored point set in $\mathbb{R}^d$. For any p, $r_p(S) \leq b_p(S)$.* □

3 Approximating MDCS

Let S be a k-colored set of points in $\mathbb{R}^d$ with L_p metric. In this section we will show that the minimum diameter rainbow set can be approximated by computing the minimum color-spanning ball. Since we can compute $b_p(S)$ efficiently, this gives us polynomial-time approximation algorithms for $\texttt{MDCS}(d, p)$.

Lemma 4. *In any dimension d with any L_p metric, the smallest color-spanning ball is an α_p^d-approximation for the minimum diameter rainbow set.*

Proof. Let R be a minimum diameter rainbow set of S. Let B be a smallest enclosing ball of R. Then, by Cor. 3,

$$b_p(S) \geq r_p(S) = diam_p(R) \geq \frac{diam(B)}{\alpha_p^d} \geq \frac{b_p(S)}{\alpha_p^d}\,.$$ □

We can therefore now focus on determining better bounds for α_p^d. We have already seen in Prop. 1 that $\alpha_1^d = \alpha_\infty^d = 1$. We can therefore solve MDCS optimally by computing a smallest color-spanning ball.

Theorem 5. *We can solve $\texttt{MDCS}(d, 1)$ and $\texttt{MDCS}(d, \infty)$ by computing a smallest color-spanning ball of S.* □

For other L_p metrics, the minimum-color spanning ball is at least a 2-approximation for the minimum diameter rainbow set. It does not necessarily minimize the diameter, as can be seen in Fig. 1.

Theorem 6. *For any d and p, $\alpha_p^d \leq 2$.*

Proof. Let B be a smallest enclosing ball of a point set R. Then there must be two points in R with distance at least $\frac{1}{2} diam_p(B)$. Otherwise, R would be completely contained in a half-ball of B, but then B would not be the smallest enclosing ball of R. □

For L_2 metric, we have a stronger bound.

Theorem 7. $\alpha_2^d \leq \sqrt{\frac{2d}{d+1}}$. □

Proof. In L_2 metric, the smallest enclosing ball B of a regular simplex in $\mathbb{R}^d$ with unit edges is a *universal cover* [6], i.e., it contains *any* point set of diameter 1. B has diameter $\sqrt{\frac{2d}{d+1}}$. □

Corollary 8. *In two dimensions with L_2 metric, we can find a $\frac{2}{\sqrt{3}} \approx 1.154$-approximation to $diam_p(S)$ in time $O(kn \log n)$. In three dimensions, we can find a $\sqrt{\frac{3}{2}} \approx 1.225$-approximation to $diam_p(S)$ in time $O(k^{1+\epsilon}n^2)$, for any $\epsilon > 1$. In higher dimensions, the approximation factor is never more than $\sqrt{2} \approx 1.414$.* □

In two dimensions, we can improve the bound for α_p^2.

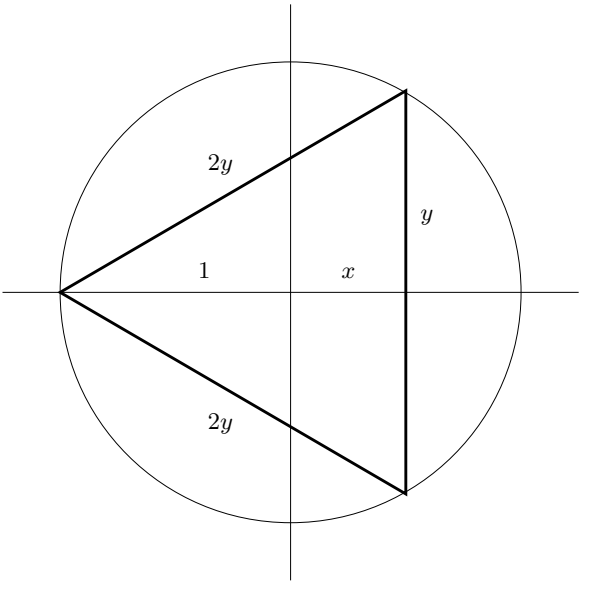


Fig. 3. An equilateral triangle with side length $2y$ in L_p metric in a ball of diameter 2

Theorem 9. *For $2 \leq p \leq \infty$, $\alpha_p^2 < \frac{4}{3}$.*

Proof. By Prop. 1, $\alpha_\infty^d = 1$, so we can assume that $2 \leq p < \infty$. It is straightforward to adapt the proof in [6] to L_p metrics to show that the smallest enclosing ball B of a regular simplex in $\mathbb{R}^d$ with unit edges is a universal cover. But we do not have a closed formula for the diameter of that ball. Fig. 3 shows a ball of diameter 2 in $\mathbb{R}^2$ with an inscribed equilateral triangle of side length $2y$ (the figure assumes L_2 metric, but the figure would be simular for any L_p metric, $2 \leq p < \infty$). We have two constraints for x and y:

$$x^p + y^p = 1 \tag{1}$$

$$\text{and} \quad (1+x)^p + y^p = (2y)^p\,. \tag{2}$$

Substituting $y^p = 1 - x^p$ in Eq. (2) yields

$$(1+x)^p + (2^p - 1)x^p = 2^p - 1\,.$$

Note that the left-hand side of this equation is monotone increasing in x. If $p \geq 2$, then it is not larger than the right-hand-side if we set $x = \frac{1}{2}$. Thus, $x \geq \frac{1}{2}$. Then, Eq. 2 implies

$$y^p = \frac{(1+x)^p}{2^p - 1} > \left(\frac{3}{4}\right)^p\,.$$

Thus, $\alpha_p^2 = \frac{1}{y} < \frac{4}{3}$. □

4 Hardness of MDCS

We do not know whether the approximation algorithms in the previous section are optimal (we suspect they are not) or whether there exists a PTAS, but we cannot hope to solve the problem exactly in polynomial time because we will now show that MDCS is NP-hard.

We first give the *decision version* of MDCS: Given an instance of MDCS and a positive number d, decide whether there exists a rainbow set of diameter at most d. Clearly, this problem is in NP (we can compare the square of all pairwise point distances to d^2, avoiding costly square root calculations). Hardness of the decision problem implies hardness of the optimization problem.

Theorem 10. *The decision version of* MDCS *is NP-hard for L_p metric, for $1 < p < \infty$, in two or higher dimensions.*

Proof. We prove the hardness of MDCS by reduction from $3SAT$. We give the proof for L_2 metric in two dimensions and then show how to extend it to other L_p metrics. This implies hardness for higher dimensions.

We first sketch the proof under the assumption that we can easily compute coordinates of points on a circle. We will then show how to approximate these coordinates with low precision rational numbers.

Let F be a Boolean formula in conjunctive normal form with n variables $x_1, \ldots, x_n$ in m clauses $c_1, \ldots, c_m$ of size at most three. To construct an instance I of MDCS, we draw a circle C with diameter 1. For each variable x_i, we define two antipodal slots s_i and $\bar{s}_i$ on C, corresponding to the positive literal x_i and the negative literal $\bar{x}_i$, respectively. These slots should be pairwise distinct, but otherwise they can be placed arbitrarily; for example, we could create $2n$ equally-spaced slots on C.

For each clause c_j we create a new color col_j. If x_i appears in c_j, we place a point of color col_j at slot s_i Similarly, if $\bar{x}_i$ appears in c_j, we place a point of color col_j at slot $\bar{s}_j$. Note that several points can coincide if a literal appears in several clauses. Finally, we set $d = 1 - \epsilon$ for some ϵ defined below. See Fig. 4 for an example of the construction.

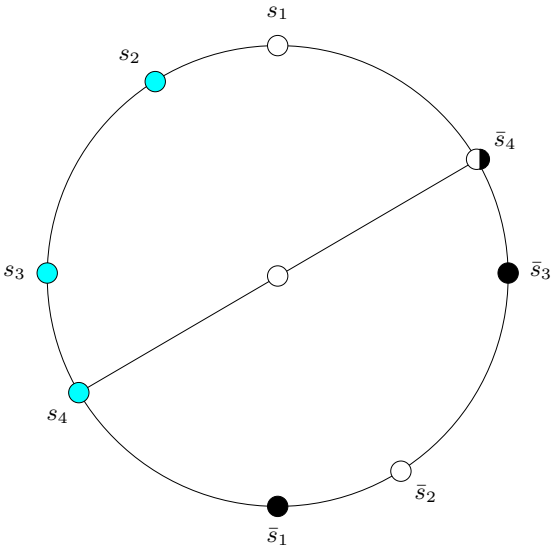


Fig. 4. An example for the construction in the NP-hardness reduction. Let $F = (x_1 \wedge \bar{x}_2 \wedge \bar{x}_4) \vee (x_2 \wedge x_3 \wedge x_4) \vee (\bar{x}_1 \wedge \bar{x}_3 \wedge \bar{x}_4)$. The literals of the first clause are colored white, the second clause colored cyan, and the third clause colored black. Note that a white and black point share slot $\bar{s}_4$.

If I has a solution, then there is a rainbow set R of m points with diameter at most $d = 1 - \epsilon$. In particular, R cannot contain both s_i and $\bar{s}_i$, for any i. Therefore, R induces a truth assignment for the variables x_i. If $s_i \in R$, we set $x_i = 1$, otherwise we set $x_i = 0$. Since R contains one point of each color, each clause will contain at least one true literal, i.e., F will be satisfied.

If F admits a satisfying assignment, then each clause c_j contains at least one true literal x_{i_j} (or $\bar{x}_{i_j}$). We add the corresponding point of color col_j at slot s_{i_j} (or $\bar{s}_{i_j}$) to the set R. Then, R is a rainbow set of diameter at most d. Thus, I has a solution.

We now discuss the problem of approximating the slot coordinates with low precision rationals. For example, if the slots are evenly spaced around the circle, we can set ϵ to be any value strictly smaller than $\frac{\pi}{n}$. If we chose $\epsilon < \frac{\pi}{2n}$, we can approximate the slot positions by choosing an arbitrary point inside a ball centered at the slot with diameter at most $\frac{\epsilon}{2}$.

For other L_p metrics, $1 < p < \infty$, observe that any two points on the boundary of a unit disc will have distance at most 1, with equality if and only if the two points are antipodal. Therefore, the proof above also works for arbitrary L_p metrics, except if $p = 1$ and $p = \infty$, since for these metrics the discs are actually squares and any two points on opposite sides of the square have distance equal to the diameter of the disc. □

5 Conclusions

We have shown that MDCS can easily be solved for L_1 and L_∞ metrics, while it is NP-hard for other L_p metrics. Unfortunately, the approximation ratio of the smallest color-spanning disc deteriorates if the dimension increases. It would therefore be interesting to find better approximation algorithms for MDCS, in particular in higher dimensions. It may also be worthwhile to study FPT algorithms for MDCS, for example with parameter k, the number of colors. The problem is clearly polynomial-time for two colors, and our NP-hardness reduction required a large number of colors.

Acknowledgments

We thank the reviewers for their helpful suggestions how to improve the presentation of the paper.

References

1. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristan, V.: The farthest color Voronoi diagram and related problems. In: Proceedings of the 17th European Workshop on Computational Geometry (EWCG 2001), pp. 113–116 (2001)
2. Abellanas, M., Hurtado, F., Icking, C., Klein, R., Langetepe, E., Ma, L., Palop, B., Sacristan, V.: Smallest color-spanning objects. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 278–289. Springer, Heidelberg (2001)
3. Chen, Y., Chen, S., Gu, Y., Hui, M., Li, F., Liu, C., Liu, L., Ooi, B.C., Yang, X., Zhang, D., Zhou, Y.: MarcoPolo: A community system for sharing and integrating travel information on maps. In: Proceedings of the 12th International Conference on Extending Database Technology (EDBT 2009), pp. 1148–1151 (2009)
4. El-Gebeily, M.A., Fiagbedzi, Y.A.: On certain properties of the regular n-simplex. International Journal of Mathematical Education in Science and Technology 35(4), 617–629 (2004)
5. Huttenlocher, D.P., Kedem, K., Sharir, M.: The upper envelope of voronoi surfaces and its applications. Discrete & Computational Geometry, 267–291 (1993)
6. Jung, H.: Über die kleinste Kugel, die eine räumliche Figur einschließt. Journal für die reine und angewandte Mathematik 123, 232–257 (1901)
7. Zhang, D., Chee, Y.M., Mondal, A., Tung, A.K.H., Kitsuregawa, M.: Keyword search in spatial databases: Towards searching by document. In: Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE 2009), pp. 688–699 (2009)

Approximation Algorithm for the Largest Area Convex Hull of Same Size Non-overlapping Axis-Aligned Squares

Wenqi Ju[1,2] and Jun Luo[1,⋆]

[1] Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, China
[2] Institute of Computing Technology, Chinese Academy of Sciences, China
{wq.ju,jun.luo}@sub.siat.ac.cn

Abstract. Given a set of n equal size and non-overlapping axis-aligned squares, we need to choose exactly one point in each square to make the area of a convex hull of the resulting point set as large as possible. Previous algorithm [10] on this problem gives an optimal algorithm with $O(n^3)$ running time. In this paper, we propose an approximation algorithm which runs in $O(n \log n)$ time and gives a convex hull with area larger than the area of the optimal convex hull minus the area of one square.

Keywords: Convex Hull, Imprecise Data, Computational Geometry.

1 Introduction

Finding a convex hull is a classic problem in computational geometry and it is used in many application domains such as pattern recognition [1], data mining [4] *etc.* There are many papers on computing the convex hull [3] [6] [8] [7] [9]. But all those classic algorithms assume that the input data are precise. In fact, data in real life, more often than not, are recorded approximately in computer because of computational error or privacy protection [2] [5] *etc.* The imprecise data can be presented by circle model or line segment model. Also the axis-aligned squares are often used to present the imprecise data.

The problem we discuss in this paper is: Given a set of n equal size and non-overlapping axis-aligned squares, we need to choose exactly one point in each square such that the area of the convex hull of the resulting point set as large as possible. Löffler *et al.* [10] propose an $O(n^3)$ algorithm for this problem. In this paper, we propose an approximation algorithm for the problem. The difference between the area of the convex hull computed by our algorithm and the area of the optimal convex hull is less than the area of one square and the time complexity is only $O(n \log n)$.

⋆ This work was supported by 2008-China Shenzhen Inovation Technology Program (SY200806300211A).

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 293–303, 2010.

2 $O(n^3)$ Algorithms by Löffler *et al.* [10]

In [10], Löffler *et al.* give some important observations. We borrow some definitions and lemmas from [10]. Let p_t, p_r, p_l and p_b, called extreme points, be the topmost, rightmost, leftmost and bottommost vertices of a convex hull. These four points divide the convex hull into four parts: top left, top right, bottom left and bottom right chains. Similarly topmost, rightmost, leftmost and bottommost square of input set are called extreme squares and denoted as S_t, S_r, S_l and S_b.

Lemma 1. *There is an optimal solution where all points lie at a corner of their square.*

Lemma 2. *All vertices on the top left chain are top left corners of their squares, and similar for the other three chains.*

Lemma 3. *An extreme square in the input set gives one of the extreme points of the optimal solution.*

The general idea of the $O(n^3)$ algorithms in [10] is: first we fix four extreme points that have constant number of combinations. Then constructing the optimal convex hull by reducing the problem to constructing the largest area convex hull on parallel line segments model by dynamic programming method which takes $O(n^3)$ time.

3 Approximation Algorithm

Let $CH(v_1, v_2, ..., v_k)$ be the convex hull such that $v_1, v_2, ..., v_k$ are the vertices in counterclockwise order. We denote the area of $CH(v_1, v_2, ..., v_k)$ by $Area(v_1, v_2, ..., v_k)$. For square S, let $S^{tl}, S^{tr}, S^{bl}, S^{br}$ be the top left, top right, bottom left and bottom right corner vertices of S respectively. Let $x.a$ and $y.a$ be the x and y coordinates of point a.

Without loss of generality, we assume the input squares are unit squares and there are only one topmost square, one bottommost square, one leftmost square and one rightmost square at most. According to lemma 1,2 and 3, there are only at most 16 combinations of extreme points. Now assuming four extreme points are fixed, we construct the convex hull as follows: use p_l, p_t and all top left corners of all non-extreme squares to construct top left chain in $O(n \log n)$ time. Other three convex chains are constructed in similar way. Therefore, we can construct at most 16 such convex hulls in $O(n \log n)$ time. We call those convex hulls as CH_{origin}.

If there is a non-extreme square of which more than one corner appear as vertices of one convex hull, we call this square *invalid square*. If we use a line l to sweep the convex hull from left to right (or from top to bottom if l is horizontal), the length of intersection line segment between l and the convex hull increases first and then decreases. We denote the increasing region as l_+ and the deceasing region l_-. In this paper, l is fixed to vertical, horizontal, positive 45 degree and negative 45 degree lines denoted as $|, -, /, \backslash$ respectively.

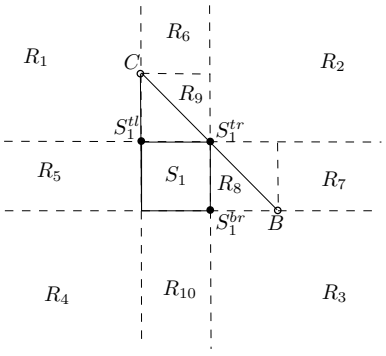


Fig. 1. It is impossible that more than two vertices of non-extreme squares are on CH_{origin}

Lemma 4. *At most two vertices of one invalid square can appear as vertices of* CH_{origin}.

Proof. Let us assume there is a non-extreme square S_1 and $S_1^{tl}, S_1^{tr}, S_1^{br}$ are on top left, top right and bottom right chains of CH_{origin} respectively. We divide the plane into 10 subdivisions from R_1 to R_{10} (see Figure 1). R_8, R_9 are unit squares on the right and the top of S_1. According to the properties of the convex hull, there is no vertex lying in R_1, R_2 and R_3. Also there is no vertex in R_8 and R_9 since the squares don't overlap. Therefore, p_r, p_t of CH_{origin} should appear in R_7, R_6 respectively. However, according to the property of convex hull, p_r, p_t have to be B, C respectively where B is the bottom right corner of R_8 and C is the top left corner of R_9. Then S_1^{tr} is on the line segment of BC. We can treat S_1^{tr} as inside the convex hull.

From lemma 4, we know there are only 6 types of invalid squares by choosing different pairs of corners to appear on CH_{origin}. For invalid square S, if S^{tl} and S^{bl} appear on CH_{origin}, we call it S_{lNear}. Similarly, other three invalid squares are denoted as S_{tNear}, S_{bNear}, S_{rNear}. However if S^{tl} and S^{br} appear on CH_{origin}, there are two cases: if the diagonal line through S^{tl} and S^{br} is on $\backslash_+$ region, then we denote the invalid square as S_{blNear}, otherwise as S_{trNear}. Similarly we define S_{tlNear}, and S_{brNear}. Therefore, we have 8 types of invalid squares totally.

Lemma 5. *There is not any part of non-extreme square to the left of left edge of* S_{lNear}*, to the right of right edge of* S_{rNear}*, to the top of top edge of* S_{tNear} *and to the bottom of bottom edge of* S_{bNear}*. There is not any part of non-extreme square to the left of left edge of* S_{blNear} *and below the bottom edge of* S_{blNear}*. Similar situations also hold for* S_{trNear}*,* S_{tlNear} *and* S_{brNear}*.*

Proof. We will prove the first case. Other three cases for $S_{rNear}, S_{tNear}, S_{bNear}$ have similar proof. Assume that there is a non-extreme square S_1 to the left of

left edge of S_{lNear}. According to lemma 2, the left edge of S_{lNear} appears on $|_+$. If we draw a line l through the left edge of S_1, since l is to the left of left edge of S_{lNear}, the length of the intersection line segment between l and CH_{origin} should be less than 1. Thus at least one of the two left corners of S_1 is out of CH_{origin} which contradicts the way we constructed CH_{origin}.

Now we prove there is not any part of non-extreme square to the left of left edge of S_{blNear}. Other cases have similar proof. Suppose there is a non-extreme square S_1 to the left of left edge of S_{blNear}. There are two subcases:

1. S_1 is below S_{blNear} (see Figure 2(a)). Let l_1 be the line through S^{tl}_{blNear} and S^{br}_{blNear}, l_2 be the line through S^{tl}_1 and S^{br}_1. Since no squares overlap and S_1 is below S_{blNear}, l_2 should be to the left of l_1. Furthermore l_1 is on the region of $\backslash_+$, then the length of the intersection line segment of l_2 with CH_{origin} is less than $\sqrt{2}$ that means at least one vertex of S^{tl}_1 and S^{br}_1 is out of CH_{origin}. That contradicts the way we construct CH_{origin}.
2. S_1 is above S_{blNear} (see Figure 2(b)). If we draw vertical and horizontal lines through S^{tl}_{blNear} and divide the plane into four regions R_1, R_2, R_3, R_4. According to the property of convex hull, there are no vertices on region R_1. Then S_1 can not appear on R_1 thus can not appear to the left of left edge of S_{blNear}.

Note that this is also true that there is not any part of non-extreme square in the left plane of the line through S^{tl}_{blNear} and S^{br}_{blNear} since no squares overlap. Similar situations also hold for S_{trNear}, S_{tlNear} and S_{brNear}.

From lemma 5, we know some pairs of invalid squares can not coexist. For example, S_{lNear} and S_{blNear} can not appear simultaneously since both require there are no non-extreme squares appearing to the left of their left edges. Thus we have the following corollary.

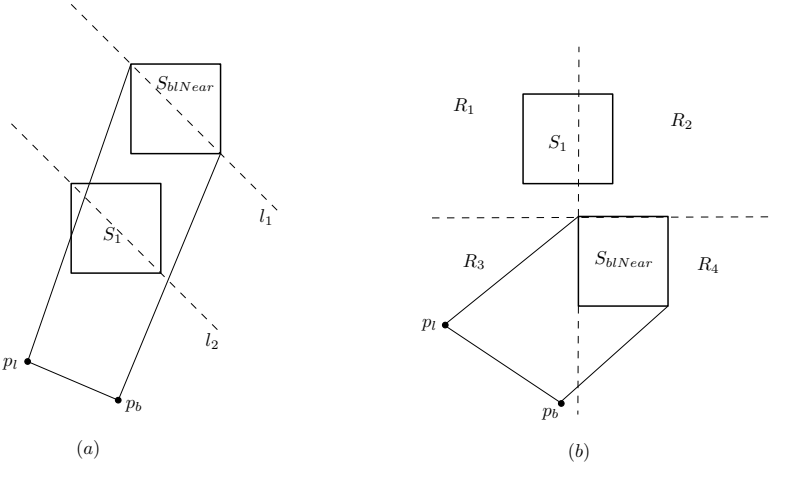


Fig. 2. Illustration for lemma 5

Corollary 1. *The following pairs of invalid squares can not coexist:* $\{S_{lNear}, S_{blNear}\}$, $\{S_{lNear}, S_{tlNear}\}$, $\{S_{rNear}, S_{brNear}\}$,$\{S_{rNear}, S_{trNear}\}$, $\{S_{tNear}, S_{tlNear}\}$, $\{S_{tNear}, S_{trNear}\}$, $\{S_{bNear}, S_{blNear}\}$, $\{S_{bNear}, S_{brNear}\}$, $\{S_{tlNear}, S_{blNear}\}$, $\{S_{blNear}, S_{brNear}\}$, $\{S_{brNear}, S_{trNear}\}$ *and* $\{S_{trNear}, S_{tlNear}\}$.

Lemma 6. *If* S_{lNear} *exists, we can move* p_l *to the other left corner of* S_l *and reconstruct the convex hull to make* S_{lNear} *valid square such that the resulting convex hull has larger area and no new invalid squares appear. Similar properties hold for* S_{rNear}, S_{bNear}, S_{tNear}.

Proof. Suppose p_l is S_l^{bl} for CH_{origin}. Since two left vertices of S_{lNear} appear on CH_{origin} and on region $|_+$, then $y.S_{lNear}^{tl} > y.S_l^{bl} > y.S_{lNear}^{bl}$. Thus $y.S_l^{tl} > y.S_{lNear}^{tl}$ since squares have same size. If we move p_l from S_l^{bl} to S_l^{tl} then reconstruct the convex hull, S_{lNear}^{tl} is inside the new convex hull. According to lemma 5, there is no square to the left of left edge of S_{lNear}. After moving p_l from S_l^{bl} to S_l^{tl}, S_{lNear}^{bl} is still the neighbor vertex of p_l on the bottom left chain of CH_{new}. Since p_r, p_b, p_t are fixed, the only vertices could be changed are from top left chain. Also because p_l is moved vertically upward, the vertices can only disappear on CH_{new} without adding new vertices and the new convex hull has larger area.

If S_{trNear} exists, there are two cases: S_t and S_r are different or same.

Lemma 7. *If* S_{trNear} *exists and* S_t *and* S_r *are different, we can move* p_t *or* p_r *and reconstruct the convex hull to make* S_{trNear} *valid square such that the area of new convex hull is at least the area of old convex hull minus* $\frac{1}{2}$ *and no new invalid squares appear. Similar properties hold for* S_{brNear}, S_{blNear}, S_{tlNear}.

Proof. Depending on $y.S_r^{tr} \leq y.S_t^{br}$ or $y.S_r^{tr} > y.S_t^{br}$, we have two cases:

1. $y.S_r^{tr} \leq y.S_t^{br}$. There are still four subcases depending on p_t and p_r from which corner of S_t and S_r:
   (a) p_t is S_t^{tr} and p_r is S_r^{tr} (see Figure 3). We can move p_t to S_t^{tl} and reconstruct the convex hull denoted as CH_{new}. First, we prove that S_{trNear} is no longer invalid in CH_{new}. Suppose S_{trNear} is still invalid. We draw a line l_1 through S_t^{tl} and S_t^{br} and a line l_2 through S_{trNear}^{tl} and S_{trNear}^{br}. Because S_r^{tr} is the rightmost extreme point, then $x.S_r^{tr} \geq x.S_t^{tr}$. If S_r^{tr} is on the right half plane of l_1 (see Figure 3(a)), because $y.S_r^{tr} \leq y.S_t^{br}$, then S_r^{tr} is on the intersection region of the the right half plane of l_1 and the bottom half plane of the line through the bottom edge of S_t. Therefore the length of intersection line segment between l_1 and CH_{new} is larger than $\sqrt{2}$. There is a contradiction because the length of line segment $S_{trNear}^{tl}S_{trNear}^{br}$ is $\sqrt{2}$, l_1 and l_2 are in region $\backslash_-$, and l_2 is to the left of l_1. If S_r^{tr} is on the left half plane of l_1 (see Figure 3(b)), because $y.S_r^{tr} \leq y.S_t^{br}$, then S_r^{tr} is on the intersection region of the the left half plane of l_1 and the right half plane of the line through the right edge of S_t. Let l_3 be the line through S_r^{tr} and parallel with l_1, l_4 be the line through S_t^{tl} and S_{trNear}^{tl} and l_5 be the line through S_t^{br} and S_{trNear}^{br}.

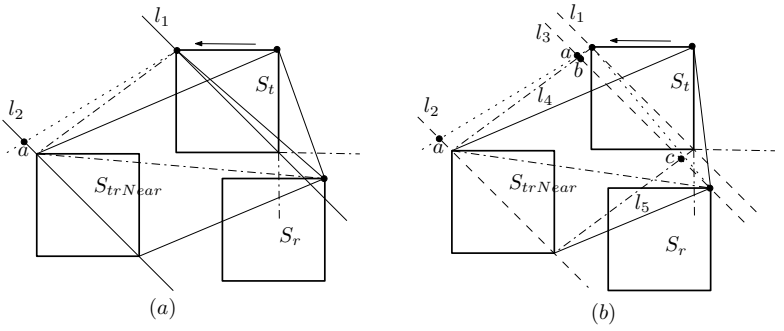


Fig. 3. Illustration for lemma 7

Let the intersection points of l_3 with CH_{new}, l_4 and l_5 be a, b and c respectively. Then the length of intersection line segment between l_3 and CH_{new} is $|aS_r^{tr}| \geq |bS_r^{tr}| \geq |bc| = \sqrt{2}$. There is a contradiction.
Second, we prove that moving p_t to S_t^{tl} does not add new vertex on CH_{new}. Since p_l, p_b, p_r are not moved, vertices can be changed only on top left chain and top right chain. After moving p_t from right to left, the vertices on top left chain can only disappear. According to lemma 5, there is not any part of non-extreme square on the left half plane of the line through S_{trNear}^{tl} and S_{trNear}^{br}. Then the only vertex could be added on top right chain is S_{trNear}^{tr}. However we can prove this will not happen. Suppose S_{trNear}^{tr} appears on the top right chain of CH_{new}. Let S_{trNear}^{bl} on $(0, 0)$ (see Figure 4). We draw a unit square S_1 just to the right of S_{trNear}. Since S_{trNear}^{tr} is on the top right chain, S_{trNear}^{br} is on the bottom right chain, and S_r can not overlap with S_{trNear}, then $0 \leq y.p_r \leq 1$ and $x.p_r \geq 2$. Let l_2 be the line through S_{trNear}^{br} and p_r, l_1 be the line through S_{trNear}^{tl} and S_t^{tr}, and l_3 be the line through S_{trNear}^{tr} and p_r. Let the intersection point of l_2 and l_3 be a. Let the angle between l_1, l_2 and horizontal line are β and α respectively. We want to prove that the vertical distance d from a to top edge of S_t is less than $1/2$. Since the length of top edge of S_t is fixed and S_t^{tl} should be on the left half plane of l_3. According to convex property, to make d as large as possible, we put S_t^{tl} on l_3. Since the negative 45 degree diagonal line of S_{trNear} is on $\backslash\_$ region, then $\alpha \geq \beta$. To make d as large as possible, we set $\beta = \alpha$. Also to make d larger, we can move p_r to left. But $x.p_r \geq 2$, so we can set $x.p_r = 2$ and $y.p_r = y^*$. The equation of line l_3 is $y-(y^*-1)x-4+y^* = 0$ and The equation of line l_2 is $y - y^*x - 1 = 0$ where $0 \leq y^* \leq 1$. Then the coordinate of a is

$$\begin{cases} x.a = 2 - y^* \\ y.a = 1 + y^* - y^{*2} = -(y^* - \frac{1}{2})^2 + \frac{5}{4} \leq \frac{5}{4} \end{cases}$$

Then the vertical distance d' from a to the top edge of S_{trNear} is $\leq 1/4$. Since top edges of S_{trNear} and S_t have the same length and are parallel,

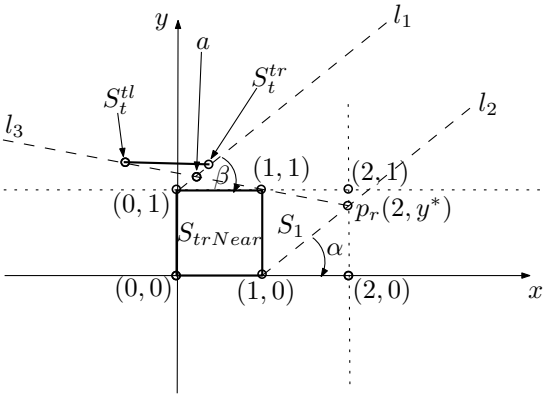


Fig. 4. Illustration for lemma 7

then $d = d' \leq 1/4 < 1/2$. Therefore the distance between top edges of S_{trNear} and S_t is $d + d' < 1$. Because S_t^{tr} and S_{rtNear}^{tl} is on the top left chain of CH_{origin}, $x.S_t^{tr} > x.S_{rtNear}^{tl}$. That means S_t overlaps S_{rtNear}. This contradicts the fact all squares are non-overlapping.
Finally, we need to prove $Area(CH_{new}) \geq Area(CH_{origin}) - 1/2$. We draw a line through S_{rtNear}^{tl} and S_{rtNear}^{br} and let it intersect with top right chain of CH_{new} at point a. Then

$$\begin{aligned}
&Area(CH_{new}) - Area(CH_{origin}) \\
&= Area(S_t^{tl}, a, S_{rtNear}^{br}, S_r^{tr}) - Area(S_t^{tr}, S_{rtNear}^{tl}, S_{rtNear}^{br}, S_r^{tr}) \\
&\geq Area(S_t^{tl}, S_{rtNear}^{tl}, S_r^{tr}) - Area(S_t^{tr}, S_{rtNear}^{tl}, S_r^{tr}) \\
&= \tfrac{1}{2}(y.S_r^{tr} - y.S_{rtNear}^{tl}) \\
&= \tfrac{1}{2}(y.S_r^{tr} - y.S_{rtNear}^{br} - 1) \\
&\geq -\tfrac{1}{2}
\end{aligned}$$

The last line of above formula is true because S_{rtNear}^{br} and S_r^{tr} are on the bottom right chain and S_r^{tr} is the rightmost point that leads to $y.S_r^{tr} - y.S_{rtNear}^{br} \geq 0$.

(b) p_t is S_t^{tl} and p_r is S_r^{tr}. This is the same situation after moving p_t in the first subcase. So no S_{rtNear} in this case.

(c) p_t is S_t^{tr} and p_r is S_r^{br}. We move p_t from S_t^{tr} to S_t^{tl}. The rest of proof is similar to the first subcase.

(d) p_t is S_t^{tl} and p_r is S_r^{br}. This is the same situation after moving p_r and p_t to make S_{rtNear} valid for the second and third subcases. So no S_{rtNear} in this subcase.

2. $y.S_t^{tr} \leq y.S_r^{tr} \leq y.S_t^{br}$. We can reduce this case to the first case. For example, if p_t is S_t^{tr} and p_r is S_r^{tr} (see Figure 5(a)), we move p_r from S_r^{tr} to S_r^{br}. We know $x.S_r^{tr} > x.S_t^{tr} + 1$ since no squares overlap. If we draw a line l through S_r^{tr} and S_r^{bl} and symmetric image over l for all squares, then this case is exactly the same as the first subcase in previous case.

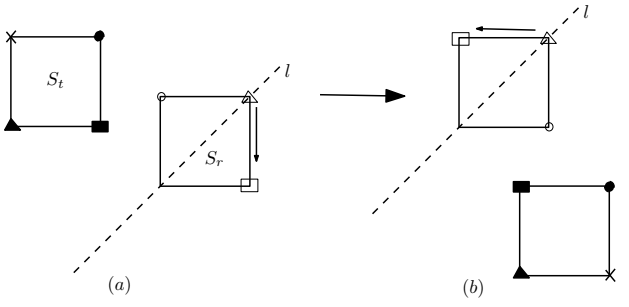


Fig. 5. (b) is symmetric image of (a) over l

Now let us discuss the situation when S_t and S_r are the same square denoted as S_{tr}. Let the topmost square except S_{tr} be S_{t*} and rightmost square except S_{tr} be S_{r*}. There are two subcases: $p_r \neq p_t$ or $p_r = p_t$.

If $p_r \neq p_t$, we could have $p_t = S^{tl}_{tr}$ or $p_r = S^{bl}_{tr}$. Since those two cases are symmetric, we only consider $p_t = S^{tl}_{tr}$.

Lemma 8. *If $p_t = S^{tl}_{tr}$ and $p_r = S^{tr}_{r*}$, we can move p_r from S^{tr}_{r*} to S^{br}_{r*} and reconstruct the convex hull to make S_{trNear} valid square such that the area of new convex hull is at least the area of old convex hull minus $\frac{1}{2}$ and no new invalid squares appear.*

Proof. First we prove S_{trNear} is no longer invalid on CH_{new}. Suppose S_{trNear} still appears on CH_{new}. Since $p_r = S^{tr}_{r*}$ and S_{r*} is the second rightmost square, we have $x.S^{tl}_{tr} \leq x.S^{tr}_{r*} = x.S^{br}_{r*} \leq x.S^{tr}_{tr}$ (see Figure 6). Let the line through S^{br}_{r*} and S^{tl}_{r*} be l_1 and the line through S^{br}_{trNear} and S^{tl}_{trNear} be l_2. The length of the intersection segment of l_1 with CH_{new} is $\geq \sqrt{2}$ since S^{tl}_{r*} is inside CH_{new}. That contradicts the fact l_1 is to the right of l_2 and they are in $\backslash\_$ region of CH_{new}.

Second, since p_t, p_l, p_b are fixed, moving p_r from S^{tr}_{r*} to S^{br}_{r*} could only change top right chain and bottom right chain. For bottom right chain, the movement only makes the vertices on bottom right chain disappear. For top right chain, the only possible new vertex could add in is S^{tr}_{trNear}. Similar to the proof of lemma 7, we can prove this could not happen.

Finally, we need to prove $Area(CH_{new}) - Area(CH_{origin}) \geq -\frac{1}{2}$. Assume l_2 intersects with CH_{new} at point a (see Figure 6).

$$
\begin{aligned}
&Area(CH_{new}) - Area(CH_{origin}) \\
&= Area(S^{tl}_{tr}, S^{tl}_{rtNear}, a, S^{br}_{r*}) - Area(S^{tl}_{tr}, S^{tl}_{rtNear}, S^{br}_{rtNear}, S^{tr}_{r*}) \\
&\geq -Area(S^{tl}_{tr}, S^{br}_{r*}, S^{tr}_{r*}) \\
&= -\tfrac{1}{2} \times 1 \times (x.S^{tr}_{r*} - x.S^{tl}_{tr}) \\
&\geq -\tfrac{1}{2}
\end{aligned}
$$

The last inequality is true because $x.S^{tl}_{tr} \leq x.S^{tr}_{r*} = x.S^{br}_{r*} \leq x.S^{tr}_{tr}$.

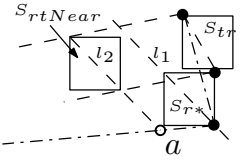


Fig. 6. Illustration for lemma 8

Lemma 9. *If $p_t = S^{tl}_{tr}$ and $p_r = S^{br}_{r*}$, then S_{rtNear} can not appear on CH_{origin}.*

Proof. This is true because this situation is the same as the situation in lemma 8 after moving p_r from S^{tr}_{r*} to S^{br}_{r*}.

Now let us discuss the situation when $p_r = p_t$. First we discuss the situation of $p_r = p_t = S^{tr}_{tr}$. There are still two subcases: S_{t*} and S_{r*} are different squares or the same square.

Lemma 10. *If $p_r = p_t = S^{tr}_{tr}$ and S_{t*} and S_{r*} are different squares, then S_{rtNear} can not appear on CH_{origin}.*

Proof. Suppose S_{rtNear} appears on CH_{origin}. According to lemma 5, there is not any part of non-extreme squares on the right half plane of the line l_1 through S^{tl}_{trNear} and S^{br}_{trNear}. Let the horizontal line through top edge of S_{trNear} be l_2. Since S_{t*} is the topmost square except S_{tr}, top edge of S_{t*} has to appear in the intersection region of the left half plane of l_1 and upper half plane of l_2. Therefore two top vertices of S_{t*} are out of CH_{origin} that contradicts the way we construct CH_{origin}. Note that if $S_{trNear} = S_{t*}$, then there is no contradiction. Similar proof for S_{r*}. Since S_{t*} and S_{r*} are different, at least one contradiction exists.

Lemma 11. *If $p_r = p_t = S^{tr}_{tr}$ and S_{t*} and S_{r*} are the same square denoted as S_{tr*}, then S_{rtNear} is S_{tr*}. We could choose $p_t = S^{tl}_{tr}$ or $p_r = S^{br}_{tr}$ and reconstruct the convex hull to make S_{trNear} valid square such that the area of new convex hull is at least the area of old convex hull minus $\frac{1}{2}$ and no new invalid squares appear.*

Proof. First we prove S_{rtNear} is S_{tr*}. Assume S_{rtNear} is not S_{tr*}. According to lemma 5, there is no non-extreme squares on the right half plane of the line l_1 through S^{tl}_{trNear} and S^{br}_{trNear}. Therefore the top edge or right edge of S_{tr*} should be to the bottom or the left of S_{trNear} that contradicts the definition of S_{tr*}.

For $S_{rtNear} = S_{tr*}$, there are three subcases:

1. The top edge of S_{tr*} is below the bottom edge of S_{tr} and the right edge of S_{tr*} is to the left of the left edge of S_{tr} (see Figure 7). In this case, we choose $p_t = p_r = S^{tl}_{tr}$ or $p_t = p_r = S^{br}_{tr}$ to construct the new convex hull. Let l_2 be the line through S^{tl}_{tr} and S^{tl}_{tr*} and l_3 be the line through S^{br}_{tr} and S^{br}_{tr*}. Suppose S_{tr*} is still invalid after set $p_t = p_r = S^{tl}_{tr}$. Then all squares

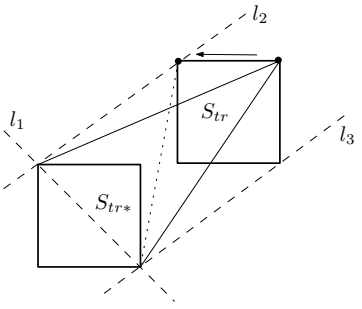


Fig. 7. Illustration for lemma 11

are on the bottom half plane of l_1. Then we set $p_t = p_r = S^{br}_{tr}$. Similarly if S_{tr*} is still invalid, all squares are on the top half plane of l_2. Since l_2 and l_3 are parallel, the centers of all squares are on the same line which can be easily handled (details are omitted). Without loss of generality we assume no centers of three squares are on the same line. Therefore, choose either $p_t = p_r = S^{tl}_{tr}$ or $p_t = p_r = S^{br}_{tr}$ will make S_{tr*} no longer invalid. Similar to the proof of lemma 7, we know there are no new vertices added on CH_{new}. For the area, we have:

$$\begin{aligned}
&Area(CH_{new}) - Area(CH_{origin}) \\
&\geq Area(S^{tl}_{tr}, S^{tl}_{tr*}, S^{br}_{tr*}) - Area(S^{tr}_{tr}, S^{tl}_{tr*}, S^{br}_{tr*}) \\
&= Area(S^{tl}_{tr}, S^{tl}_{tr*}, S^{tr}_{tr}) - Area(S^{tl}_{tr}, S^{br}_{tr*}, S^{tr}_{tr}) \\
&= -\tfrac{1}{2} \times 1 \times (y.S^{tl}_{tr*} - y.S^{br}_{tr*}) \\
&= -\tfrac{1}{2}
\end{aligned}$$

2. The top edge of S_{tr*} is below the bottom edge of S_{tr} and the right edge of S_{tr*} is to the right of the left edge of S_{tr}. In this case, we set $p_t = S^{tl}_{tr}$ and $p_r = S^{br}_{tr*}$. This is similar the situation of lemma 9 and there is no S_{trNear} on CH_{new}. For the area, we have exactly the same formula as the first subcase.
3. The top edge of S_{tr*} is above the bottom edge of S_{tr} and the right edge of S_{tr*} is to the left of the left edge of S_{tr}. This case is symmetric to the second subcase.

If $p_r = p_t = S^{tl}_{tr}$ or $p_r = p_t = S^{br}_{tr}$, from above proof, we know one of them has no S_{trNear}. Then for the one has S_{trNear}, we can move $p_r = p_t$ to the other diagonal vertex to make S_{trNear} disappear.

Theorem 1. *For n unit non-overlapping axis-aligned squares, we have $O(n \log n)$ time approximation algorithm to find the largest area convex hull such that exactly one point in each square is chosen to construct the convex hull. The area of the approximate convex hull is at most 1 less than the area of the optimal convex hull.*

Proof. The 16 CH_{origin} can be constructed in $O(n \log n)$ time. We choose the largest area convex hull CH^{max}_{origin} from those 16 CH_{origin}. If one invalid square

appears, according to lemma 6 to lemma 11, we could move extreme vertices to make CH_{new} such that $Area(CH_{new}) \geq Area(CH^{max}_{origin}) - \frac{1}{2}$. Also we know from lemma 6 to lemma 11 that invalid squares causing the area decrease at most $\frac{1}{2}$ by moving vertices can only be S_{brNear}, S_{blNear}, S_{tlNear} and S_{trNear}. According to corollary 1, at most two of those four invalid squares could exist simultaneously. So the area could be decreased at most by 1. Therefore $Area(CH_{new}) \geq Area(CH^{max}_{origin}) - 1 \geq Area(CH_{opt}) - 1$ where CH_{opt} is the optimal convex hull. CH_{new} can be constructed in $O(n \log n)$ time.

4 Conclusions

In this paper, we present an approximation algorithm for calculating the largest area convex hull with the model of the same size squares. The running time is $O(n \log n)$ and the difference between the area of our result and the area of the optimal convex hull is no more than the area of one square. The algorithm is suitable for those realtime applications which have limited computing time. For the future works, it will be interesting to investigate the approximate algorithms for the overlapping squares by using our methods.

References

1. Akl, S.G., Toussaint, G.T.: Efficient convex hull algorithms for pattern recognition applications. In: Int. Joint Conf. on Pattern Recognition, Kyoto, Japan, pp. 483–487 (1978)
2. Beresford, A.R., Stajano, F.: Location Privacy in pervasive Computing. IEEE Pervasive Computing 2(1) (2003)
3. Bhattacharya, B.K., E1 Gindy, H.: A new linear convex hull algorithm for simple polygons. IEEE Trans. Inform. Theory IT-30, 85–88 (1984)
4. Böhm, C., Kriegel, H.: Determing the convex hull in large multidimensional databases. In: Kambayashi, Y., Winiwarter, W., Arikawa, M. (eds.) DaWaK 2001. LNCS, vol. 2114, pp. 294–306. Springer, Heidelberg (2001)
5. Gedik, B., Liu, L.: A customizable k-anonymity model for protecting location privacy. In: ICDCS (2005)
6. Graham, R.L.: An efficient algorithm for determining the convex hull of a finite planar set. Information Processing Letters 26, 132–133 (1972)
7. Graham, R.L., Yao, F.F.: Finding the convex hull of a simple polygon. J. Algorithms 4, 324–331 (1984)
8. Lee, D.T.: On finding the convex huff of a simple polygon. Internat. J. Comput. Inform. Sci. 12, 87–98 (1983)
9. Melkman, A.: On-Line Construction of the Convex Hull of a Simple Polyline. Information Processing Letters 25, 11–12 (1987)
10. Loffler, M., van Kreveld, M.: Largest and Smallest Convex Hulls for Imprecise Points. Algorithmica 56(2), 235–269 (2010)

Optimum Sweeps of Simple Polygons with Two Guards

Xuehou Tan[1,2] and Bo Jiang[1]

[1] Dalian Maritime University, Linghai Road 1, Dalian, China
[2] Tokai University, 4-1-1 Kitakaname, Hiratsuka 259-1292, Japan
tan@wing.ncc.u-tokai.ac.jp

Abstract. A polygon P admits a *sweep* if two mobile guards can detect an unpredictable, moving target inside P, no matter how fast the target moves. For safety, two guards are required to always be mutually visible, and thus, they should move on the polygon boundary. Our objective in this paper is to find an optimum sweep such that the sum of the distances travelled by the two guards in the sweep is minimized. We present an $O(n^2)$ time and $O(n)$ space algorithm, where n is the number of vertices of the given polygon. This new result is obtained by converting the problem of sweeping simple polygons with two guards into that of finding a shortest path between two nodes in a graph of size $O(n)$.

1 Introduction

Motivated by the relations to the well-known *Art Gallery* and *Watchman Route* problems, much attention has recently been devoted to the problem of detecting an unpredictable, moving target in an n-sided polygon P by a group of mobile guards. Both the target and the guards are modeled by points that can continuously move in P. The goal of the guards is to eventually "see" the target, or to verify that no target is present in the polygon, no matter how fast the target moves. Many types of polygon shapes and the vision sensors of the guards have been studied in the literature [3,5,6,7,8,10,11,13,14].

In this paper, we focus on the two-guard model studied in [3,6], in which two guards move on the polygon boundary and are always kept to be mutually visible. The goal is to sweep P with two guards so that at any instant, the line segment connecting the guards partitions P into a "clear" region (not containing the target) and an "uncleared" region (it may contain the target). In the end, we would like to know whether the whole polygon P is clear or the target is detected, if it is ever possible. In generally, one can consider to sweep a polygonal region with a chain of $k+1$ guards (k is a positive integer) such that consecutive guards along the chain are mutually visible. For instance, efficient algorithms for computing the minimum number r^* of guards required to sweep P as well as for generating a sweep schedule have been presented in [3,13]. This

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 304–315, 2010.

target-finding model may have applications in adversarial settings, as it has obvious advantages for safety and communication between guards.

Icking and Klein were the first to study the problem of sweeping corridors with two guards, which was called the *two-guard problem* [6]. A simple polygon P with an *entrance* s and an *exit* t on its boundary is called a *corridor*. The problem of sweeping corridors with two guards asks the guards to start at the entrance s and force the target out of the region through the exit t. They gave an $O(n \log n)$ time algorithm for determining whether a corridor can be swept with two guards [6]. Later, a linear time algorithm was presented by Heffernan [5]. Tseng *et al.* gave an $O(n \log n)$ time algorithm to determine whether there is a pair of vertices in P that allows a sweep. This result has recently been improved to $O(n)$ [1]. If a corridor can be swept with two guards, a sweep schedule consisting of the minimum number m of sweep instructions can be reported in $O(n \log n + m)$ time. Note that m has a lower bound $\Omega(n^2)$ [6].

The problem of sweeping simple polygons with two guards was later studied [13,14]. Since neither the entrance nor the exit on the polygon boundary is given, the starting point (on the polygon boundary) of any sweep schedule may be visited by the target for the second or more time, i.e., *recontamination* is generally necessary for the problem of sweeping simple polygons with a chain of guards [3,4,14]. This makes it more difficult and challenge. The previous research was mainly concentrated on determining whether a sweep is possible for the given polygon and reporting a sweep schedule (if it exists). For instance, a linear time algorithm for determining whether a polygon can be swept with two guards and a quadratic algorithm for reporting a sweep schedule have been presented in [13,14].

Our objective in this paper is to find an optimum sweep such that the sum of the distances travelled by the two guards in the sweep is minimized. The motivation for studying this problem (called the *min-sum* problem) arises from the fact that the cost or energy required by a mobile robot (guard) is an increasing function of the distance it travelled.

In Section 2 of this paper, we give basic definitions empolyed in the paper [5,6]. It has been known that a sweep of the given polygon can be decomposed into a sequence of two basic motions of the guards (called the *straight/counter sweeps* in this paper), in which both guards never change their moving directions. Moreover, the starting/ending position of a basic motion is given by a ray-shooting segment (which connects a reflex vertex and one of its ray shots) [6,14]. In Section 3, we present a through study of all basic sweeps among the ray-shooting segments. In Section 4, we introduce a data structure that records the ray-shooting segments (i.e., the starting/ending positions of basic motions) and a transition relation among all basic motions of the two guards. By applying Dijkstra's algorithm to the obtained diagram, we can then give our $O(n^2)$ time and $O(n)$ space algorithm for finding an optimum sweep of simple polygons such that the sum of the travelled distances is minimized. In Section 5, we show that the $O(n^2)$ time solution to the min-sum problem is optimal in the worst case.

2 Preliminary

Let P denote a simple polygon of n vertices and ∂P the boundary of P. Two points p, $q \in P$ are *visible* from each other if the line segment connecting them, denoted by $\overline{pq}$, does not intersect the exterior of P.

Let g_1, g_2 be two point guards on ∂P. Let $g_1(t)$ and $g_2(t)$ denote the positions of g_1 and g_2 on ∂P at time t; we require that $g_1(t)$ and $g_2(t)$: $[0, \infty) \rightarrow \partial P$ be two continuous functions. A point $x \in P$ is said to be *detected* or *illuminated* at t if x is contained in the line segment $\overline{g_1(t)g_2(t)}$. A *configuration* of g_1 and g_2 at time t is a pair of the points $g_1(t)$ and $g_2(t)$ such that the line segment $\overline{g_1(t)g_2(t)}$ lies in the interior of P. Assume that the initial positions of two guards are located at a vertex or on an edge of P. The configuration of g_1 and q_2 at a time thus divides P into a *clear* region that does not contain the target, and an uncleared (or a contaminated) region that may contain the target.

As in [6], a *sweep instruction* can be given by a pair of functions $g_1(t)$, $g_2(t)$ such that either of $g_1(t)$ and $g_2(t)$ specifies an algebraic path, i.e., an edge of P along which the guard g_1 or g_2 moves. More specifically, the following two types of sweep instructions are considered: Two guards g_1 and g_2 move along segments of single edges such that (i) no intersections occur among all line segments $\overline{g_1(t)g_2(t)}$ during the movement, or (ii) any two segments $\overline{g_1(t)g_2(t)}$ intersect each other. See Figs. 1(a)-(b), where the clear region of the polygon P is shaded. (Probably, one guard stands still, while the other moves.)

Let the area of P be one, and let $P(t)$ denote the fraction of the clear area at time t. Clearly, $P(0) = 0$. We say a *sweep schedule* exists for P, or equally, P allows a *sweep* if $P(t) = 1$ for some $t > 0$. The *complexity* of a sweep schedule is the total number of sweep instructions it consists of.

For a vertex v of the polygon P, let $Succ(v)$ denote the vertex immediately succeeding v *clockwise*, and $Pred(v)$ the vertex immediately preceding v *clockwise*. A vertex of P is *reflex* if its interior angle is strictly greater than 180°. An important definition for reflex vertices is that of *ray shots*: the backward ray shot from a reflex vertex r, denoted by *Backw(r)*, is the first point of P hit by a "bullet" shot at r in the direction from $Succ(r)$ to r, and the forward ray shot *Forw(r)* is the first point hit by the bullet shot at r in the direction from

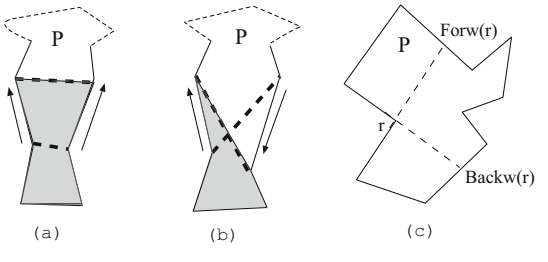


Fig. 1. Sweep instructions, and ray shots

$Pred(r)$ to r. See Fig. 1(c). The vertex r is referred to as the *origin* of the shots $Backw(r)$ and $Forw(r)$. The segment $\overline{rBackw(r)}$ or $\overline{rForw(r)}$ is referred to as the *ray-shooting segment*.

3 Basic Sweeps among the Ray-Shooting Segments

It has been known that a sweep of the given polygon can be decomposed into a sequence of two basic motions of the guards, in which only instructions (i), or only instructions (ii) are used [5,6,14]. Moreover, the starting/ending position of a basic motion is given by a ray-shooting segment. A sweep between two ray-shooting segments is said to be *straight* if only instructions (i) are used, or *counter* if only instructions (ii) are used. Clearly, the two ray-shooting segments for straight (resp. counter) sweeps are disjoint (resp. intersect each other).

3.1 Straight Sweeps

We will first review the structure of the restrictions placed on the motion of the two guards in straight sweeps, which was originally studied for the problem of sweeping corridors with two guards [5,12], and then, show how it can be used in the solution to the problem of sweeping simple polygons with two guards.

Let a and b denote two endpoints of the current segment connecting the two guards, which separates the clear region from the contaminated region. Without loss of generality, assume that the region right to the segment $\overline{ab}$, as viewed from a, is clear. Assume below that all boundary points of P are ordered clockwise, starting at the point a. Let p_1, p_2 be the two reflex vertices such that their backward ray-shooting segments $\overline{p_1Backw(p_1)}$ and $\overline{p_2Backw(p_2)}$ are contained in the contaminated region. Consider a straight sweep from $\overline{ab}$, which is devoted to clearing both $\overline{p_1Backw(p_1)}$ and $\overline{p_2Backw(p_2)}$. The shot $Backw(p_2)$ is said to be *dominated*, with respect to $\overline{ab}$, if p_1 precedes p_2, and $Backw(p_1)$ precedes $Backw(p_2)$. See Figs. 2(a)-(b). As discussed in [5,12], the shot $Backw(p_1)$ (resp. $Backw(p_2)$) in Fig. 2(a) imposes a requirement that one guard should reach $Backw(p_1)$ (resp. $Backw(p_2)$) by the time the other reaches an internal point of $\overline{p_1Succ(p_1)}$ (resp. $\overline{p_2Succ(p_2)}$) in the straight sweep. Analogously, the symmetric situation in Fig. 2(b) imposes a requirement that one guard should not leave $Backw(p_1)$ (resp. $Backw(p_2)$) before the other leaves the edge $\overline{p_1Succ(p_1)}$ (resp. $\overline{p_2Succ(p_2)}$) in the straight sweep. Clearly, the requirement imposed by $Backw(p_1)$ implies that by $Backw(p_2)$ in either situation. Hence, the shot $Backw(p_2)$ can be ignored when a straight sweep from $\overline{ab}$ that clears $\overline{p_1Backw(p_1)}$ and $\overline{p_2Backw(p_2)}$ is considered.

We can also define five other types of dominated shots between the intersecting ray-shooting segments, which are contained in the contaminated region [5,12].

- $Forw(p_1)$ is dominated with respect to $\overline{ab}$ if there exists a vertex p_2 such that p_1 precedes p_2, and $Forw(p_1)$ precedes $Forw(p_2)$ (Figs. 2(c)-(d)).
- $Backw(p_2)$ is dominated with respect to $\overline{ab}$ if there exists a vertex p_1 such that p_1 precedes $Backw(p_2)$, and $Backw(p_1)$ precedes p_2 (Fig. 2(e)).

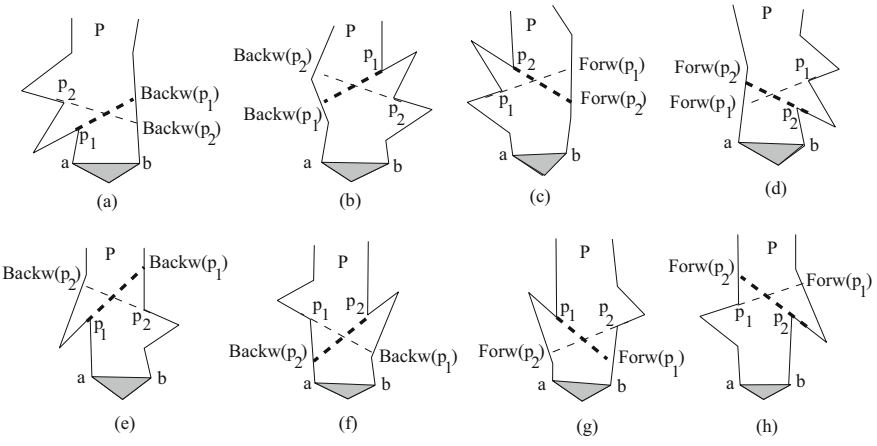


Fig. 2. Illustration for the definition of dominated shots

- $Backw(p_1)$ is dominated with respect to $\overline{ab}$ if there exists a vertex p_2 such that $Backw(p_2)$ precedes p_1, and p_2 precedes $Backw(p_1)$ (Fig. 2(f)).
- $Forw(p_2)$ is dominated with respect to $\overline{ab}$ if there exists a vertex p_1 such that $Forw(p_2)$ precedes p_1, and p_2 precedes $Forw(p_1)$ (Fig. 2(g)).
- $Forw(p_1)$ is dominated with respect to $\overline{ab}$ if there exists a vertex p_2 such that p_1 precedes $Forw(p_2)$, and $Forw(p_1)$ precedes p_2 (Fig. 2(h)).

A shot is said to be *non-dominated* with respect to $\overline{ab}$, if it is not dominated by any other shots. Clearly, each family of non-dominated shots has the *non-crossing* property [5,14]. For example, if the origins of non-dominated backward shots are in clockwise order, $p_1, \ldots, p_k$ on ∂P, then their shots $Backw(p_1), \ldots, Backw(p_k)$ are in counterclockwise order on ∂P.

The concept of non-dominated shots is used in our solution to the problem of sweeping simple polygons with two guards. The idea here is to consider all possible straight sweeps from the current segment $\overline{ab}$. We claim that it suffices to compute at most two straight sweeps from $\overline{ab}$ to the ray-shooting segments in the contaminated region. Denote by a_1 the first reflex vertex, which succeeds a and whose ray-shooting segment $\overline{a_1 Backw(a_1)}$ is contained in the contaminated region. Clearly, $Backw(a_1)$ dominates all the backward shots $Backw(p_2)$ shown in Fig. 2(a). Similarly, denote by $Backw(b_1)$ the first backward shot, which succeeds a and whose ray-shooting segment $\overline{b_1 Backw(b_1)}$ is wholly contaminated. Also, $Backw(b_1)$ dominates all the shots $Backw(p_1)$ shown in Fig. 2(b). If the dominance relation between $Backw(a_1)$ and $Backw(b_1)$ occurs (Figs. 2(e)-(f)), only the straight sweep towards the ray-shooting segment with the non-dominated shot is possible; in this case, we determine whether this straight sweep really exists. Otherwise, two segments $\overline{a_1 Backw(a_1)}$ and $\overline{b_1 Backw(b_1)}$ are disjoint; in this case, only the straight sweep towards the ray-shooting segment which is closer to $\overline{ab}$ need be considered, and thus, we determine whether this straight sweep

really exists. (Since the same procedure is done for all ray-shooting segments, whether the straight sweep between $\overline{a_1 Backw(a_1)}$ and $\overline{b_1 Backw(b_1)}$ exists will be determined, when $\overline{a_1 Backw(a_1)}$ or $\overline{b_1 Backw(b_1)}$ is considered as the starting segment $\overline{ab}$.) For the forward non-dominated shots with respect to $\overline{ab}$, the same treatment is done analogously. (Further consideration between the forward and backward non-dominated shots is not needed, as it can leave to the decisions on the existences of the two straight sweeps from $\overline{ab}$.) Hence, our claim is proved.

In order to solve the problem of sweeping the given polygon with two guards, we perform the above operation, for every ray-shooting segment, by taking as two different starting segments $\overline{ab}$ (i.e., either of its endpoints is considered as the point a once). For every vertex v, we also consider $\overline{vv}$ as a ray-shooting segment, and perform the above operation for the starting segment $\overline{vv}$ *once*. Note that the straight sweep from $\overline{vv}$ with the wholly contaminated polygon P represents the very first motion of the two guards, and on the other hand, the straight sweep towards $\overline{vv}$ gives the very last motion of the two guards. We say a straight sweep between two ray-shooting segments is *canonical* if it is reported in the above procedure.

Let us now consider the time required to find all canonical straight sweeps. First, one can compute all backward and forward ray shots in $O(n \log n)$ time using the ray-shooting query algorithm [2]. For straight sweeps, the following result is known in the literature.

Lemma 1. *[5] Suppose that all ray shots inside the polygon P have been precomputed in $O(n \log n)$ time. Given two disjoint, internal segments with their endpoints on ∂P, one can determine in linear time whether a straight sweep between them exists.*

For a starting segment $\overline{ab}$, we can find in linear time two reflex vertices a_1 and b_1, as described above. It then needs to determine whether a straight sweep from $\overline{ab}$ exists at most twice. Hence, we have the following result.

Lemma 2. *All canonical straight sweeps in P can be computed in $O(n^2)$ time.*

3.2 Counter Sweeps

A *counter* sweep is a sweep in which both guards move on ∂P clockwise (or counterclockwise), in such a way that they are always mutually visible. Also, a counter sweep depends on the structure of ray shots. Again, let a, b denote the two endpoints of the current ray-shooting segment, which separates the clear region from the contaminated region. Assume also that the region right to the segment $\overline{ab}$ (viewed from a) is clear, and all boundary points of P are ordered clockwise, starting at a. Consider a counter sweep from $\overline{ab}$, which is devoted to clearing a pair of disjoint ray-shooting segments having a clear endpoint and a contaminated endpoint. See Fig. 3. Analogously, we can define the following six types of *c-dominated* shots [5,12].

- $Backw(p_1)$ is *c*-dominated with respect to $\overline{ab}$ if there exists a vertex p_2 such that p_1 precedes p_2, and $Backw(p_2)$ precedes $Backw(p_1)$ (Figs. 3(a)-(b)).

- $Forw(p_2)$ is c-dominated with respect to $\overline{ab}$ if there exists a vertex p_1 such that $Forw(p_2)$ precedes $Forw(p_1)$, and p_1 precedes p_2 (Figs. 3(c)-(d)).
- $Forw(p_2)$ is c-dominated with respect to $\overline{ab}$ if there exists a vertex p_1 such that $Forw(p_2)$ precedes p_1, and $Backw(p_1)$ precedes p_2 (Fig. 3(e)).
- $Backw(p_1)$ is c-dominated with respect to $\overline{ab}$ if there exists a vertex p_2 such that p_1 precedes $Forw(p_2)$, and p_2 precedes $Backw(p_1)$ (Fig. 3(f)).
- $Forw(p_1)$ is c-dominated with respect to $\overline{ab}$ if there exists a vertex p_2 such that $Backw(p_2)$ precedes p_1, and $Forw(p_1)$ precedes p_2 (Fig. 3(g)).
- $Backw(p_2)$ is c-dominated with respect to $\overline{ab}$ if there exists a vertex p_1 such that p_1 precedes $Backw(p_2)$, and p_2 precedes $Forw(p_1)$ (Fig. 3(h)).

A shot is said to be *non-c-dominated*, with respect to $\overline{ab}$, if it is not c-dominated by any other shots. Clearly, each family of non-c-dominated shots has the *crossing* property [5]. For an example, if $p_1, \ldots, p_k$ are ordered clockwise on ∂P, their non-c-dominated shots $Backw(p_1), \ldots, Backw(p_k)$ are ordered clockwise on ∂P.

The concept of non-c-dominated shots is also used in the solution to the problem of sweeping simple polygons with two guards. Again, it suffices to consider at most two counter sweeps from $\overline{ab}$. Denote by a_1 the last reflex vertex, which is contaminated but whose shot $Backw(a_1)$ is clear. Clearly, $Backw(a_1)$ c-dominates all the shots $Backw(p_1)$ shown in Fig. 3(a). Denote by $Forw(b_1)$ the last forward shot, which is contaminated but whose origin b_1 is clear. Also, $Forw(b_1)$ c-dominates all the shots $Forw(p_2)$ shown in Fig. 3(c). If the c-dominance relation occurs between $Backw(a_1)$ and $Forw(b_1)$ (Figs. 3(e)-(f)), only the counter sweep towards the ray-shooting segment with the non-c-dominated shot is possible; in this case, we determine whether this counter sweep really exists. Otherwise, $\overline{a_1 Backw(a_1)}$ and $\overline{b_1 Backw(b_1)}$ intersect each other; in this case, three considered segments (including $\overline{ab}$) are pairwisely intersected, and thus, we only need to determine whether the counter sweep towards the ray-shooting segment, whose contaminated endpoint is closer to a in clockwise

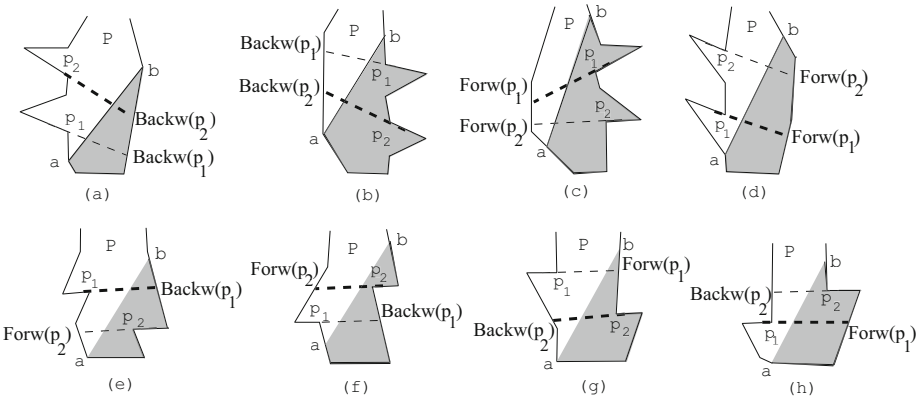


Fig. 3. Definition of c-dominated shots

direction, really exists. For the situations shown in Fig. 3(b), Fig. 3(d) and Figs. 3(g)-(h), we also need to determine whether a counter sweep from $\overline{ab}$ is possible.

For every ray-shooting segment, we perform the above operation by taking it as the starting segment $\overline{ab}$ *twice*. Also, we say a counter sweep between two ray-shooting segments is *canonical* if it ever is reported in this procedure. Analogous to the complexity analysis made for canonical straight sweeps, we have the following results.

Lemma 3. *[5] Suppose that all ray shots inside the polygon P have been precomputed in $O(n \log n)$ time. Given two intersecting, internal segments with their endpoints on ∂P, one can determine in linear time whether a counter sweep between them exists.*

Lemma 4. *All canonical counter sweeps in P can be computed in $O(n^2)$ time.*

4 Algorithm for the Min-sum Problem

In this section, we introduce a data structure, called the *ray-shooting segment diagram*, which records the ray-shooting segments and a transition relation among all basic motions of the two guards. Next, we show that any non-trivial sweep can be represented by a path between two special nodes of the ray-shooting segment diagram. This makes it possible to apply Dijkstra's algorithm to the ray-shooting segment diagram, so as to find an optimum solution.

Suppose that all vertices of P and the ray shots are ordered on ∂P clockwise. For simplicity, we number all vertices and ray shots in the sorted order using integers 0, 1, ..., $m-1$ ($m < 3n$). The ray-shooting segment diagram G is constructed as follows. First, we put into the set $V(G)$ all the nodes (i, j) ($0 \leq i, j \leq m-1$), where one of i and j is a ray shot and the other is its origin. We call these nodes (i, j) the *segment-nodes*. Without loss of generality, assume that a sweep schedule starts (resp. ends) at a vertex. So for every vertex k, we put the node (k, k) into $V(G)$. The nodes (k, k) are considered as the possible starting points of sweep schedules, and thus, called the *starting vertex-nodes*. Moreover, we put a copy of (k, k), denote by (k', k') ($k' = k$), into $V(G)$. The nodes (k', k') are considered as the possible ending points of sweep schedules, and called the *ending vertex-nodes*. Finally, we add two special nodes s (called the *source* node), t (called the *target* node) to $V(G)$. The total number of nodes of $V(G)$ is clearly less than $4n$.

Consider how to construct the arc set of the diagram G, which is denoted by $E(G)$. Two nodes A, B of $V(G)$ are connected by two symmetric arcs (in opposite directions) if and only if both A and B are the segment-nodes, and there is a canonical straight or counter sweep between A and B. Let C denote a vertex-node, and let D denote a segment-node. The node C is connected to D by a *single* arc from C to D if and only if there is a canonical straight sweep from C to D. Analogously, C is connected to D by a *single* arc from D to C if and only if there is a canonical straight sweep from D to C. Note that the very first

or last motion of any sweep schedule is a straight sweep (that starts or ends at a polygon vertex). Finally, for every starting vertex-node (k, k), we add an arc from s to (k, k). And, for every ending vertex-node (k', k'), we add an arc from (k', k') to t. Since at most two canonical straight (resp. counter) sweeps from a ray-shooting segment are examined, the total number of arcs of the obtained set $E(G)$ is $O(n)$. Observe that the transition relation among canonical straight and counter sweeps is implicitly represented by all pairs of the nodes adjacent in the diagram G, which give all arcs of the set $E(G)$.

Lemma 5. *The ray-shooting segment diagram G of a simple polygon can be constructed in $O(n^2)$ time and $O(n)$ space.*

Proof. First, all ray shots can be computed in $O(n \log n)$ time using the ray-shooting query algorithm [2]. Moreover, all canonical straight and counter sweeps can be found in $O(n^2)$ time (Lemmas 3 and 4). Since the number of nodes of $V(G)$ is no more than $4n$ and the size of $E(G)$ is $O(n)$, the lemma simply follows. □

Following from the definitions of non-dominated and non-c-dominated shots, any sweep schedule can be transformed, by deleting all unnecessary instructions, into the one that consists of only canonical sweeps. A sweep schedule is said to be *non-trivial* if it consists of only canonical straight and counter sweeps. Then, we have the following result.

Lemma 6. *The polygon P can be swept with two two guards if and only if the diagram G contains a directed st-path.*

Proof. Assume first that P can be swept with two guards. Fix a non-trivial sweep schedule $\mathcal{S}$. Then, $\mathcal{S}$ can be decomposed into a sequence of canonical straight and counter sweeps. Following from our construction of the diagram G, all canonical straight and counter sweeps are represented by arcs of G. Hence, the configuration of two guards in a canonical sweep of $\mathcal{S}$ at any time instant can be mapped to the arc representing that canonical sweep. (Remember that a canonical sweep between two ray-shooting segments are represented by two symmetric arcs in G.) An infinite number of the configurations of the guards may correspond to the same arc. Thus, the sweep schedule $\mathcal{S}$ can be mapped to a sequence of arcs in G. Next, add two additional arcs to the obtained sequence; one connects s to the first node of the arc sequence representing $\mathcal{S}$, and the other connects the last node of the arc sequence. Since $\mathcal{S}$ gives a continuous motion of the two guards, the resulting sequence is a directed st-path in G.

Assume now that G contains a directed st-path. For any arc of the st-path, we can transform it into a canonical sweep of the two guards inside the polygon P. Notice that the first two arcs of the st-path represent a straight sweep that gives the very first clear region, and the last two arcs represent another straight sweep that clears the whole polygon P. The st-path in G can thus be transformed into a sweep schdule of the two guards. □

The following observation is simple (and necessary to our algorithm).

Observation 1. *Suppose that the sum of the distances travelled by the two guards is minimized in a sweep schedule $\mathcal{S}$. Then, $\mathcal{S}$ is non-trivial.*

To solve the min-sum problem, we assign a weight with each arc of $E(G)$. First, the weight of an arc connected to the node s or t is defined as zero. Since all other arcs of $E(G)$ represent canonical sweeps, the weight of an arc is defined as the sum of the distances travelled by the two guards in the canonical sweep. We denote by G_s this weighted diagram of G.

It then follows from Lemma 6 as well as Observation 1 that a min-sum directed st-path in G_s corresponds to an optimum sweep of the polygon P. Note that whether sweeping P with two guards is possible can be determined using the graph G_s or G, too. Since a min-sum st-path in G_s can be computed using Dijkstra's algorithm, we conclude:

Theorem 1. *Given a simple polygon P, one can compute in $O(n^2)$ time and $O(n)$ space an optimum sweep of P, if it exists, such that the sum of the distances travelled by the two guards is minimized.*

5 A Lower Bound

We show that the starting point of any sweep of the polygon P may be recontaminated, and a sweep schedule can consist of $\Omega(n^2)$ sweep instructions. So our $O(n^2)$ time algorithm for the min-sum problem is optimal in the worst case.

We need more definitions. Let x, y denote two boundary points of P, and $\partial P[x, y]$ (resp. $\partial P(x, y)$) the closed (resp. open) *clockwise* chain of ∂P from x to y. A pair of reflex vertices u, v is said to form a *p-deadlock*, $p \in \partial P$, if both $\partial P(u, Backw(u)]$ and $\partial P[Forw(v), v)$ do not contain p, and they intersect each other [6,14].

Lemma 7. *There exists a polygon P such that the starting point of any sweep of P has to be recontaminated, and any sweep schedule consists of $\Omega(n^2)$ sweep instructions.*

Proof. The polygon P depicted in Fig. 4 can be swept with two guards. Fig. 4(a) gives the first straight sweep starting at s, and Fig. 4(b) shows the clear region after several counter sweeps are performed. Fig. 4(c) gives a sweep from $\overline{bc}$ to $\overline{hi}$, which consists of several straight and counter sweeps. The whole polygon can further be cleared by some counter sweeps (symmetric to Fig. 4(b)) and the last straight sweep ending at t (symmetric to Fig. 4(a)).

We now claim that the starting point of any sweep of P has to be recontaminated. Since all points of $\partial P[a, d] \cup \partial P[g, j]$ have their deadlocks, the starting point of any sweep schedule $\mathcal{S}$ belongs to $\partial P(j, a) \cup \partial P(d, g)$; otherwise, $\mathcal{S}$ is *trivial* since a beginning portion of $\mathcal{S}$ can simply be deleted from it. Since $\partial P[b, c]$ and $\partial P[h, i]$ are disjoint, any sweep schedule has to start at a point of $\partial P(j, a)$ (resp. $\partial P(d, g)$) but end at a point of $\partial P(d, g)$ (resp. $\partial P(j, a)$). See also [14].

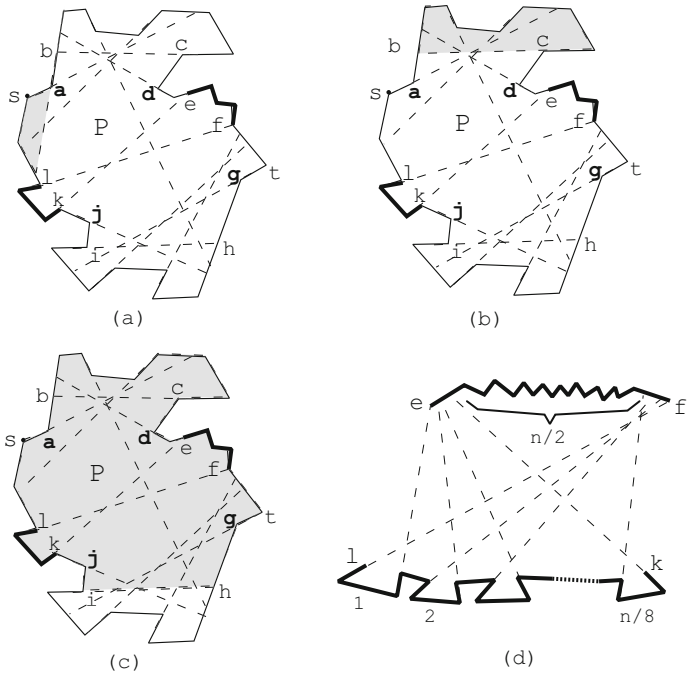


Fig. 4. Illustration for the proof of Lemma 7

Moreover, in order to obtain a clear region that contains $Pred(c)$ and $Pred(i)$ simultaneously, the sweep between $\overline{bc}$ and $\overline{hi}$ (e.g., Fig. 4(c)) is always needed. The starting point of any sweep schedule is thus contained in the contaminated region at the beginning of the sweep between $\overline{bc}$ and $\overline{hi}$ (see Fig. 4(b)). Hence, our claim is proved.

Finally, if one replaces the chains $\partial P[e, f]$ and $\partial P[k, l]$ with two long chains shown in Fig. 4(d), then the guard moving on $\partial P[e, f]$ has to visit more than $n^2/16$ edges in the sweep between $\overline{bc}$ and $\overline{hi}$ (as there are $n/8$ *wedges* in $\partial P[k, l]$ [6]). Since a sweep instruction is defined by a pair of the edges on which the two guards move, any sweep schedule for P thus consists of $\Omega(n^2)$ instructions. The proof is complete. □

References

1. Bhattacharya, B.K., Mukhopadhyay, A., Narasimhan, G.: Optimal algorithms for two-guard walkability of simple polygons. In: Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 2001. LNCS, vol. 2125, pp. 438–449. Springer, Heidelberg (2001)
2. Chazelle, B., Guibas, L.: Visibility and intersection problem in plane geometry. Discrete Comput. Geom. 4, 551–581 (1989)

3. Efrat, A., Guibas, L.J., Har-Peled, S., Lin, D.C., Mitchell, J.S.B., Murali, T.M.: Sweeping simple polygons with a chain of guards. In: Proc., ACM-SIAM Sympos. Discrete Algorithms, pp. 927–936 (2000)
4. Guibas, L.J., Latombe, J.C., Lavalle, S.M., Lin, D., Motwani, R.: Visibility-based pursuit-evasion in a polygonal environment. IJCGA 9, 471–493 (1999)
5. Heffernan, P.J.: An optimal algorithm for the two-guard problem. IJCGA 6, 15–44 (1996)
6. Icking, C., Klein, R.: The two guards problem. IJCGA 2, 257–285 (1992)
7. LaValle, S.M., Simov, B., Slutzki, G.: An algorithm for searching a polygonal region with a flashlight. IJCGA 12, 87–113 (2002)
8. Lee, J.H., Park, S.M., Chwa, K.Y.: Searching a polygonal room with one door by a 1-searcher. IJCGA 10, 201–220 (2000)
9. Park, S.M., Lee, J.H., Chwa, K.Y.: Visibility-based pursuit-evasion in a polygonal region by a searcher. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 456–468. Springer, Heidelberg (2001)
10. Suzuki, I., Yamashita, M.: Searching for mobile intruders in a polygonal region. SIAM J. Comp. 21, 863–888 (1992)
11. Tan, X.: A characterization of polygonal regions searchable from the boundary. In: Akiyama, J., Baskoro, E.T., Kano, M. (eds.) IJCCGGT 2003. LNCS, vol. 3330, pp. 200–215. Springer, Heidelberg (2005)
12. Tan, X.: The two-guard problem revisited and its generalization. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 847–858. Springer, Heidelberg (2004)
13. Tan, X.: Sweeping simple polygons with the minimum number of chain guards. Inform. Process. Lett. 102, 66–71 (2007)
14. Tan, X., Jiang, B.: Searching a polygonal region by two guards. J. Comput. Sci. Tech. 23(5), 728–739 (2008)

Adaptive Algorithms for Planar Convex Hull Problems*

Hee-Kap Ahn[1] and Yoshio Okamoto[2]

[1] Department of Computer Science and Engineering,
Pohang University of Science and Technology, Korea
heekap@postech.ac.kr
[2] Graduate School of Infomation Science and Engineering,
Tokyo Institute of Technology, Japan
okamoto@is.titech.ac.jp

Abstract. We study problems in computational geometry from the viewpoint of adaptive algorithms. Adaptive algorithms have been extensively studied for the sorting problem, and in this paper we generalize the framework to geometric problems. To this end, we think of geometric problems as permutation (or rearrangement) problems of arrays, and define the "presortedness" as a distance from the input array to the desired output array. We call an algorithm *adaptive* if it runs faster when a given input array is closer to the desired output, and furthermore it does not make use of any information of the presortedness. As a case study, we look into the planar convex hull problem for which we discover two natural formulations as permutation problems. An interesting phenomenon that we prove is that for one formulation the problem can be solved adaptively, but for the other formulation no adaptive algorithm can be better than an optimal output-sensitive algorithm for the planar convex hull problem.

1 Introduction

One can think of computational geometry as a generalization of numerical problems (namely, 1-dimensional problems) to higher dimensions. A typical example is the 2-dimensional convex hull computation, which can be thought of as a 2-dimensional generalization of sorting an array of numbers.

This work is motivated by a thorough treatment for sorting problems to take "presortedness" into account in the analysis of the algorithms. In certain cases one expects sorting algorithms to run faster if a given input is almost sorted. Mehlhorn [23] introduced the term "adaptive sorting algorithms" for those with such a property. A formal framework for the worst-case analysis of adaptive sorting algorithms was introduced by Mannila [22], and the framework is well surveyed by Estivill-Castro and Wood [15].

An adaptive sorting algorithm has several characteristics. First, it runs faster if the presortedness is high. Second, the algorithm does not use any information of the presortedness. That is a reason why it is called "adaptive."

* Work by Ahn was supported by the National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies Development. Work by Okamoto was supported by Global COE Program "Computationism as a Foundation for the Sciences" and Grant-in-Aid for Scientific Research from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 316–326, 2010.

In this paper, we study *adaptive computational geometry*. To apply the adaptiveness framework to geometric problems, we want to think of the problems as permutation problems. That is, we are given an array of objects (points, segments, etc.), and we output a permutation (or a rearrangement) of the objects that represents the desired answer. Naturally, the sorting problem is such a permutation problem, and the planar convex hull problem can be seen as a permutation problem (and actually, a lower bound of the convex hull algorithm is given by a reduction from the sorting problem). Indeed, this work is also motivated by recent studies on in-place geometric algorithms that treat some geometric problems as permutation problems [7,9,10].

Several "presortedness" measures have been proposed [15]. In this work, we use the oldest and the most frequently used measure: the number of inversions. Given two linear orders $\leq_1, \leq_2$ on X, an *inversion* is an ordered pair $(i, j) \in X^2$ such that $i <_1 j$ and $j <_2 i$. We denote the total number of inversions for $\leq_1, \leq_2$ by $\mathsf{inv}(\leq_1, \leq_2)$, Note that $\mathsf{inv}(\leq_1, \leq_2) = 0$ if and only if the order $\leq_1$ conforms to the order $\leq_2$, and therefore we may regard the number of inversions as an appropriate measure of the presortedness.

As a case study, we consider the planar convex hull computation: given a set of points in the plane, we want to compute its convex hull. For this problem, we discover two natural formulations as permutation problems. In both formulations, we require the points on the boundary of the convex hull to be sorted in clockwise order, but they are different in the treatment of the points in the interior. In the first formulation the interior points are required to be sorted (by their x-coordinates) while in the second formulation the interior points are not required so. Interestingly, this makes a huge difference in terms of complexity. We show that in the first formulation the problem can be solved in $\mathrm{O}(n(1 + \log(1 + k)))$ time when k is the number of inversions in a given array of n points with respect to the desired output. Since $k \leq \binom{n}{2}$, the running-time bound can be as bad as $\mathrm{O}(n \log n)$. Hence, this is still worst-case optimal with respect to n. On the other hand, in the second formulation we give a lower bound of $\Omega(n \log h)$ for computing the convex hull, where h is the complexity of the convex hull. This shows that the second formulation does not allow us to design any adaptive algorithm. This kind of phenomenon has not been seen for any problems for which the adaptive framework was applied.

A natural question arises here: since the convex hull of n points in the plane can be computed in linear time once they are sorted along a line or around a point, why do we need another adaptive algorithm other than an optimal adaptive sorting algorithm? An answer to the question is that any existing adaptive sorting method does not reflect the presortedness of a point set on its convex hull: consider an input array A of n points whose i-th element is a point with coordinates $(i, (-1)^i\sqrt{i})$, for $1 \leq i \leq n$. Clearly the points are already sorted along the x-axis, but not along its convex hull. As a result, A has no inversion on sorting, but has $\Omega(n^2)$ inversions on convex hull.

Related work. There are a huge number of articles discussing the adaptive sorting problem. We recommend the readers to consult a survey by Estivill-Castro and Wood [15]. Adaptive sorting algorithms are also discussed in terms of integer sorting [25] and I/O-efficiency (both cache-aware and cache-oblivious) [8].

There are several papers which apply the adaptiveness framework to problems other than sorting. Demaine, López-Ortiz, and Munro [11] considered some set operations on

sorted sets, and gave adaptive algorithms with respect to a certain measure of difficulty of the problem. The problem has been motivated from a database application, and this line of research was followed by some subsequent papers [4,3].

Levcopoulos, Lingas, and Mitchell [19] were the first to study a geometry problem in the adaptive framework. They studied the convex hull computation of a (possibly self-intersecting) piecewise linear chain. Their consideration relies on the fact that the convex hull can be computed in linear time when the chain does not have a self-intersection. Since the x-monotone non-self-intersecting chain can be seen as a sorted sequence, their study is a generalization of the adaptive sorting framework. However, they did not look at the problem as a permutation problem. Besides, Baran and Demaine [1] and Barbay and Chen [2] studied other geometric problems. However, their adaptiveness framework does not look at the presortedness and is completely different from the viewpoint of this work. In this sense, this paper studies the most fundamental counterpart of the adaptive sorting problem in computational geometry.

Notation. An array A of n elements is indexed by $1, \ldots, n$. The i-th element of A is denoted by $A[i]$, $i \in \{1, \ldots, n\}$. The subarray of A consisting of $A[i], \ldots, A[j]$ is denoted by $A[i..j]$. For a subset A' of A, the difference $A \setminus A'$ denotes the array consisting of the elements of $A \setminus A'$ and ordered as in A. The concatenation of two arrays A and B (in this order) is denoted by $A \circ B$. For a set (or an array) P of points, we denote by $\mathrm{conv}(P)$ the convex hull of P, and by $\partial\mathrm{conv}(P)$ the boundary of $\mathrm{conv}(P)$.

Weak orders. For the investigation of geometric problems, it is convenient to extend the framework for linear orders to weak orders. In general, a binary relation $\lesssim$ on a set X is a *weak order* on X if it is reflexive ($x \lesssim x$ for all $x \in X$), transitive ($x \lesssim y$ and $y \lesssim z$ imply $x \lesssim z$ for all $x, y, z \in X$), and total ($x \lesssim y$ or $y \lesssim x$ for all $x, y \in X$). We say $x \sim y$ if $x \lesssim y$ and $y \lesssim x$, and $x < y$ if $x \lesssim y$ and not $x \sim y$. Note that a weak order is a linear order if and only if it is also antisymmetric. In other words, $x \sim y$ does not necessarily imply $x = y$ for a weak order $\lesssim$. Given two weak orders $\lesssim_1, \lesssim_2$ on X, an *inversion* is an ordered pair $(i, j) \in X^2$ such that $i <_1 j$ and $j <_2 i$. We denote the total number of inversions for $\lesssim_1, \lesssim_2$ by $\mathsf{inv}(\lesssim_1, \lesssim_2)$.

2 Planar Convex Hulls

Informally speaking, in the *planar convex hull problem*, we are given a set P of n points in general position (i.e., no three points of P are collinear, and no two points have the same x-coordinate), and want to identify the points on the boundary of the convex hull of P. To cast the problem into the adaptive framework, we introduce the following two formulations.

2.1 First Formulation: The Interior Points Need to Be Sorted

We are given P as an array of n points in the plane. The output is a rearrangement Q of the array P in the following way. If h is the number of points on $\partial\mathrm{conv}(P)$, then $Q[1..h]$ should be the array of these points sorted clockwise with $Q[1]$ being the leftmost point

in P. Then, $Q[h{+}1..n]$ should be the array of points lying in the interior of $\mathrm{conv}(P)$, sorted by their x-coordinates.

Thus, we obtain two linear orders. The first order $\leq_P$ is defined by the input array P as $P[i] \leq_P P[j]$ if and only if $i \leq j$. The other order $\leq_Q$ is defined by the output array Q as $Q[i] \leq_Q Q[j]$ if and only if $i \leq j$. For these we may define the number of inversions.

2.2 Second Formulation: The Interior Points Need Not to Be Sorted

In the first formulation, it would be unnatural to require the interior points to be sorted because we are often interested in the points on the boundary of the convex hull only. Therefore, we consider the following variation. Given an array P of n points in the plane, the output is a rearrangement Q of the array P in the following way. If h is the number of points on $\partial\mathrm{conv}(P)$, then $Q[1..h]$ should be the array of these points sorted clockwise with $Q[1]$ being the leftmost point in P. Then, $Q[h{+}1..n]$ is any rearrangement of points lying in the interior of $\mathrm{conv}(P)$. So the output array Q is not uniquely determined from P. Note that this formulation has already been proposed in the literature on in-place algorithms [10].

Then, we define the following two weak orders. The first one $\leq_P$ is the same as the first formulation: $P[i] \leq_P P[j]$ if and only if $i \leq j$. The other order $\lesssim_Q$ is defined from an output array Q as $Q[i] \lesssim_Q Q[j]$ if and only if $0 \leq i \leq j \leq n$ or $h{+}1 \leq j \leq i \leq n$. That means the interior points are indifferent in $\lesssim_Q$. Note that the order $\lesssim_Q$ does not depend on a particular choice of an output Q. This is the spot where we need a weak order since the output is not determined uniquely from the input.

2.3 Results

With these two formulations, we prove the following results.

- For the first formulation, we give an adaptive algorithm running in $\mathrm{O}(n(1+\log(1+k)))$ time where $k = \mathsf{inv}(\leq_P, \leq_Q)$. We also give a lower bound $\Omega(n(1+\log(1+k/n)))$ for the number of comparisons even if the number h of points on the boundary of the convex hull is constant.
- For the second formulation, we prove that any (fixed-degree) algebraic computation tree solving the problem has height at least $\Omega(n \log h)$. Therefore, with the second framework we cannot beat an optimal output-sensitive algorithm (running in $\mathrm{O}(n \log h)$ time, e.g. [18]) and cannot obtain any adaptive algorithm.

3 An Adaptive Algorithm for the First Formulation

When designing adaptive convex hull algorithms, we may encounter at least the following two difficulties. First, we cannot determine whether $p <_Q q$ just by looking at two points p, q. It depends on how the other points are placed around p, q. Second, related to the first one, if we want to proceed by divide-and-conquer and obtain a subset P' of

P that yields the output Q', then it is not generally the case that Q' is a subarray of Q; i.e., subsets do not inherit the linear order. These two issues do not arise in the sorting problem, where any two numbers can be compared just by looking at them, and any smaller subsets inherit the order.

Our algorithm below overcomes these issues and is shown to be adaptive. The call to ConvexHull(P) identifies the upper chain $U(P)$ of $\mathrm{conv}(P)$ in the increasing order of x-coordinates, the lower chain $L(P)$ of $\mathrm{conv}(P)$ in the decreasing order of x-coordinates, and the set $I(P)$ of points in the interior of $\mathrm{conv}(P)$ in the increasing order of x-coordinates. The desired output is the concatenation $U(P) \circ (L(P) \setminus \{\text{the rightmost point, the leftmost point}\}) \circ I(P)$.

Algorithm: ConvexHull(P)

Step 1: If P is arranged as a desired output, then identify $U(P), L(P), I(P)$ and halt.

Step 2: Otherwise, compute a vertical bisector s of P. Let P_A be the set of points in P left to s, and P_B be the set of points in P right to s.

Step 3: Compute the upper common tangent u and the lower common tangent ℓ of $\mathrm{conv}(P_A)$ and $\mathrm{conv}(P_B)$. Let $a_u \in P_A$ and $b_u \in P_B$ be the two points spanning u. Similarly, let $a_\ell \in P_A$ and $b_\ell \in P_B$ be the two points spanning ℓ. Let P_L be the set of points in P that lie left to the line spanned by a_u, a_ℓ, P_R be the set of points in P that lie right to the line spanned by b_u, b_ℓ, and $P_M = P \setminus (P_L \cup P_R)$. Note that $P_L \subseteq P_A$ and $P_R \subseteq P_B$.

Step 4: Call ConvexHull(P_L) and ConvexHull(P_R) to obtain $U(P_L)$, $L(P_L)$, $I(P_L)$, $U(P_R)$, $L(P_R)$, $I(P_R)$. Identify $U(P) = U(P_L) \circ U(P_R)$ and $L(P) = L(P_R) \circ L(P_L)$.

Step 5: Sort the points in P_M by their x-coordinates to obtain the sorted sequence $S(P_M)$, and apply the merge sort for $I(P_L), S(P_M), I(P_R)$. Identify the result as $I(P)$. Halt.

The algorithm is similar to the divide-and-conquer algorithm by Kirkpatrick and Seidel [18]. In their algorithm the upper hull and the lower hull are computed separately, but we cannot do so here since we may lose the adaptiveness. Rather, we compute the upper and lower hulls simultaneously. The correctness of the algorithm is straightforward.

Now we estimate the running time. From now on, denote by n the number of input points in P, and by k the number of inversions between $\leq_P$ and $\leq_Q$.

Step 1 can be executed in $\mathrm{O}(n)$ time as follows. First we find the leftmost point p and the rightmost point q of P in $\mathrm{O}(n)$ time. In the desired output, p should come first ($p = P[1]$) and q should come somewhere, say at the h'-th position ($q = P[h']$). Then we check whether the subarray $P[1..h']$ is a concave chain C_u by looking at turns at all three consecutive points. This can be done in $\mathrm{O}(n)$ time. Next, we compute the second leftmost point p' on the lower hull of P in $\mathrm{O}(n)$ time, and determine $h \geq h'$ such that $p' = P[h]$. Then, we check whether the subarray $P[h'..h]$ with p forms a convex chain C_ℓ in $\mathrm{O}(n)$ time. Now the points $P[h{+}1..n]$ must lie in the interior of $\mathrm{conv}(P)$, and be sorted by their x-coordinates. First we check if they are sorted in $\mathrm{O}(n)$ time. Then, for each point $r \in P[h{+}1..n]$ from left to right, we check if r lies between

the concave chain C_u and the convex chain C_ℓ. This can be done in $O(n)$ time since they are all sorted. Finally, we identify $U(P) = P[1..h']$, $L(P) = P[h'..h] \circ P[1]$, $I(P) = P[h+1..n]$. Thus, we can execute Step 1 in $O(n)$ time.

Step 2 reduces to the median finding problem, which can be solved in $O(n)$ time [6].

In Step 3, computing the upper and the lower tangents reduces to 2D linear programming (as in Kirkpatrick and Seidel [18]), which can be solved in $O(n)$ time [24]. Also it is straightforward to find P_L and P_R in $O(n)$ time. Note that $|P_L|, |P_R| \leq n/2$.

Step 4 involves recursive calls. A crucial observation is that a point of P on $\partial\mathrm{conv}(P)$ lies on $\partial\mathrm{conv}(P_L)$ or $\partial\mathrm{conv}(P_R)$, and a point of P_L (and P_R) on $\partial\mathrm{conv}(P_L)$ (and $\partial\mathrm{conv}(P_R)$ respectively) lies on $\partial\mathrm{conv}(P)$. Therefore, the desired output Q_L for ConvexHull(P_L) and the desired output Q_R for ConvexHull(P_R) are subsequences of Q. This way, we succeed in overcoming the difficulties described before. If we denote by $t(n,k)$ the worst-case running time of ConvexHull(P) over all P with $|P| = n$ and $\mathrm{inv}(\leq_P, \leq_Q) = k$ (when Q is the desired output), Step 4 takes at most $t(|P_L|, k_L) + t(|P_R|, k_R)$ time, where k_L, k_R denote the number of inversions for P_L, P_R and (the restriction to P_L, P_R of) Q, respectively. Since P_L, P_R, P_M are disjoint subsequences of P, we have the following lemma.

Lemma 1. *Denote by k_L, k_R, k_M the number of inversions for P_L, P_R, P_M and (the restriction to P_L, P_R, P_M of) Q, respectively. Then, it holds that $k_L + k_R + k_M \leq k$.*

In Step 5, we sort the points in P_M. If we apply an adaptive sorting algorithm, say by Estivill-Castro and Wood [14], we can sort P_M in $O(|P_M|(1+\log(1+k_M)))$ time. Further, the merging can be done in $O(n)$ time in a standard way since $I(P_L), I(P_R), S(P_M)$ are all sorted.

Now we analyze the overall running time summarizing the discussion above. Consider all linear-time processing in Steps 1, 2, 3, 5 takes an time for some constant a and for all sufficiently large n, and the adaptive sorting in Step 5 takes $b|P_M|(1+\log_2(1+k_M))$ time for some constant b and for all sufficiently large n. If we denote the number of points in P_L, P_R, P_M by n_L, n_R, n_M, respectively (note that $n_L + n_R + n_M = n$), then we obtain the following recurrence:

$$t(n,k) \leq an + t(n_L, k_L) + t(n_R, k_R) + bn_M(1 + \log_2(1 + k_M))$$

for sufficiently large n and $k \geq 1$. Note that for small n it holds $t(n,k) = O(1)$ and when $k \leq 0$ it holds that $t(n,k) = O(n)$. We now derive that $t(n,k) \leq cn(1+\log_2(1+k))$ for some constant c and for all sufficiently large n.

By induction, we obtain

$$\begin{aligned} t(n,k) \leq\ & an + cn_L(1 + \log_2(1 + k_L)) + cn_R(1 + \log_2(1 + k_R)) \\ & + bn_M(1 + \log_2(1 + k_M)). \end{aligned}$$

We choose c so that it satisfies $2b \leq c$. Let $n_L = \alpha n$ and $n_R = \beta n$. Then we have $0 \leq \alpha \leq 1/2, 0 \leq \beta \leq 1/2$, and $n_M = (1-\alpha-\beta)n$. Note that α and β are parameters that cannot be freely chosen but depend on the input. The recurrence becomes

$$\begin{aligned} t(n,k) &\le an + c\alpha n(1+\log_2(1+k_L)) + c\beta n(1+\log_2(1+k_R)) \\ &\quad + \frac{c}{2}(1-\alpha-\beta)n(1+\log_2(1+k_M)) \\ &= an + c(\alpha+\beta+\frac{1-\alpha-\beta}{2})n \\ &\quad + cn\log_2(1+k_L)^{\alpha}(1+k_R)^{\beta}(1+k_M)^{(1-\alpha-\beta)/2} \\ &\le an + cn + cn\log_2(1+k_L)^{\alpha}(1+k_R)^{\beta}(1+k_M)^{(1-\alpha-\beta)/2}. \end{aligned}$$

Here, we want to know when the argument of the last logarithm $(1+k_L)^{\alpha}(1+k_R)^{\beta}(1+k_M)^{(1-\alpha-\beta)/2}$ is maximized. Taking the logarithm further, we reduce this maximization to the following linear program with two variables α, β:

$$\begin{aligned} &\text{maximize} \quad \alpha\log_2(1+k_L) + \beta\log_2(1+k_R) + \frac{1-\alpha-\beta}{2}\log_2(1+k_M) \\ &\text{subject to} \quad 0 \le \alpha, \beta \le 1/2. \end{aligned}$$

This problem can be directly solved. We have four cases. Let $A = \log_2(1+k_L) - \frac{1}{2}\log_2(1+k_M)$ (that is the coefficient of α), and $B = \log_2(1+k_R) - \frac{1}{2}\log_2(1+k_M)$ (that is the coefficient of β).

Case 1: when $A \ge 0$ and $B \ge 0$. Then, the optimum is attained at $\alpha = \beta = 1/2$, and the optimal value is $(\log_2(1+k_L) + \log_2(1+k_R))/2$.

Case 2: when $A \ge 0$ and $B < 0$. Then, the optimum is attained at $\alpha = 1/2, \beta = 0$, and the optimal value is $(2\log_2(1+k_L) + \log_2(1+k_M))/4$.

Case 3: $A < 0$ and $B \ge 0$. Then, the optimum is attained at $\alpha = 0, \beta = 1/2$, and the optimal value is $(2\log_2(1+k_R) + \log_2(1+k_M))/4$.

Case 4: $A < 0$ and $B < 0$. Then, the optimum is attained at $\alpha = \beta = 0$, and the optimal value is $(\log_2(1+k_M))/2$.

For each of these four cases, we proceed with the estimation of $t(n,k)$. When Case 1 occurs, we obtain

$$\begin{aligned} t(n,k) &\le (a+c)n + cn\log_2(1+k_L)^{1/2}(1+k_R)^{1/2} \\ &\le (a+c)n + cn\log_2\frac{(1+k_L)+(1+k_R)}{2} \\ &\le (a+c)n + cn\log_2\frac{2+k}{2} \le (a+c)n + cn\log_2(\frac{3}{4}(1+k)) \\ &= (a+c)n + (\log_2\frac{3}{4})cn + cn\log_2(1+k), \end{aligned}$$

where we use a relation of arithmetic means and geometric means in the second inequality, Lemma 1 in the third inequality, and $\frac{2+k}{2} \le \frac{3}{4}(1+k)$ for $k \ge 1$ in the second to last inequality. Thus, if we choose c so that it satisfies $a + (1+\log_2\frac{3}{4})c \le c$, then we obtain $t(n,k) \le cn(1+\log_2(1+k))$ as desired. Note that $\log_2\frac{3}{4} \approx -0.415037$.

When Case 2 occurs, we obtain

$$\begin{aligned} t(n,k) &\le (a+c)n + cn\log_2(1+k_L)^{1/2}(1+k_M)^{1/4} \\ &\le (a+c)n + cn\log_2(1+k_L)^{1/2}(1+k_M)^{1/2}, \end{aligned}$$

and we then go along the same line as Case 1. Case 3 is also similar. When Case 4 occurs, we obtain

$$t(n,k) \le an + cn + cn\log_2(1+k_M)^{1/2} \le an + cn + cn\log_2(1+k_M/2)$$

and we proceed with the same argument. This way, we conclude $t(n,k) \le cn(1+\log_2(1+k))$ for all of the four cases. We summarize the discussion in the following theorem.

Theorem 1. *The algorithm* ConvexHull *above computes the convex hull of a given (non-degenerate) point set in the plane in* $O(n(1+\log(1+k)))$ *time, where* n *is the number of input points and* k *is the number of inversions as defined in the first formulation, namely* $k = \mathsf{inv}(\le_P, \le_Q)$.

As for the lower bound it is easy to observe the following.

Theorem 2. *Any algorithm to solve the planar convex hull problem in the first formulation needs at least* $\Omega(n(1+\log(1+k/n)))$ *comparisons in the worst case even if the number of points on the boundary of the convex hull is constant (four). Here,* n *is the number of input points and* k *is the number of inversions as defined in the first formulation.*

Proof. We reduce the adaptive sorting problem to our problem. Guibas, McCreight, Plass, and Roberts [16] showed that any sorting algorithm needs at least $\Omega(n(1+\log(1+k/n)))$ comparisons in the worst case, where n is the size of an input array and k is the number of inversions between the positions (or indices) in the input array and the increasing order of numbers themselves.

Let A be an input array of size n for the sorting problem. Then, we construct an array P of $n+4$ planar points as follows. For each number $A[i]$ in the input array, we set $P[i{+}4] = (A[i], 0)$. This determines $P[5..n{+}4]$. The other four points are determined as follows. Let ℓ be the smallest number in A, and u be the largest number in A. Then, we set $P[1] = (\ell-2, 1), P[2] = (u+2, 1), P[3] = (u+1, -1), P[4] = (\ell-1, -1)$. This completes the construction of the point set P, and we consider the planar convex hull problem when the input array is P. Let Q be the output array. Then we can see that $P[1], \ldots, P[4]$ are the points on the boundary of the convex hull, and $k = \mathsf{inv}(\le_P, \le_Q)$. Furthermore, from Q we can extract the sorted sequence in the increasing order as the x-coordinates of $Q[5..n{+}4]$. Since ℓ and u can be found in linear time, this finishes the whole reduction. □

4 Lower Bound for the Second Formulation

To obtain a lower bound for the second formulation, we consider the following No Inversion Problem: Given an array P of (non-degenerate) point set in the plane, we want to determine whether $\mathsf{inv}(\le_P, \lesssim_Q) = 0$. The following theorems show that this problem is as hard as the planar convex hull problem itself.

Theorem 3. *Any (fixed-degree) algebraic decision tree solving the* No Inversion Problem *has height at least* $\Omega((n-h)\log h)$*, where* n *is the number of input points and* h *is the number of points on the boundary of the convex hull.*

Proof. We construct a linear-time reduction to NO INVERSION PROBLEM from the following CHIR PROBLEM:[1] Given a regular convex h-gon R with its smallest circumscribing disk D, and $n-h$ points in D, determine whether all of these $n-h$ points lies in R. Kapoor and Ramanan [17] proved that any (fixed-degree) algebraic decision tree solving the CHIR Problem has height at least $\Omega((n-h)\log h)$.[2]

For the reduction, we are given a regular convex h-gon R with its smallest circumscribing disk D, and a set X of $n-h$ points in D. Then, we construct an array P of points that is supposed to be an instance of the NO INVERSION PROBLEM as follows. At $P[1..h]$ we place the vertices of R in the clockwise order in such a way that $P[1]$ will be the leftmost one. Then, at $P[h+1..n]$ we place the points of X arbitrarily. We can see that P can be constructed in linear time. We can also see that $\mathsf{inv}(\leq_P, \lesssim_Q) = 0$ if and only if all points of X lie in R. Thus, the reduction is completed. □

Theorem 4. *Any (fixed-degree) algebraic decision tree solving the* NO INVERSION PROBLEM *has height at least* $\Omega(n \log n)$*, where* n *is the number of input points.*

Proof. Follow the proof of Theorem 3, but this time we set $h = n/2$ in the CHIR PROBLEM. Then, the same argument gives a desired lower bound. □

The following corollary is straightforward from the theorems above.

Corollary 1. *Any (fixed-degree) algebraic decision tree solving the* NO INVERSION PROBLEM *has height at least* $\Omega(n \log h)$*, where* n *is the number of input points and* h *is the number of points on the boundary of the convex hull.*

Proof. When $h < n/2$, we have an $\Omega(n \log h)$ lower bound from Theorem 3. When $h \geq n/2$, we have an $\Omega(n \log h)$ lower bound from Theorem 4. □

As shown in the following theorem, the lower bound for the NO INVERSION PROBLEM can be translated to the lower bound for the planar convex hull problem in the second formulation.

Theorem 5. *For the (fixed-degree) algebraic computation tree model, any algorithm to solve the planar convex hull problem in the second formulation requires at least* $\Omega(n \log h)$ *time, where* n *is the number of input points and* h *is the number of points on the boundary of the convex hull.*

Proof. Let A be an algorithm to solve the planar convex hull problem in the second formulation, and let Q be an output array from A when we input P into A. From Q, we can determine h in $\mathrm{O}(n)$ time as Step 1 of the algorithm CONVEXHULL in the previous section. Therefore, by looking through $P[1..h]$ and $Q[1..h]$, we can determine whether $\mathsf{inv}(\leq_P, \lesssim_Q) = 0$ in $\mathrm{O}(h)$ time. In this way, we can solve NO INVERSION PROBLEM, and so A needs at least $\Omega(n \log h)$ comparisons by Corollary 1 (note that decision by one comparison can be implemented as a node of an algebraic decision tree). □

[1] This is the abbreviation of "Convex hull inclusion with restriction" [17].

[2] The CHIR problem by Kapoor and Ramanan [17] is a bit different from ours, but the lower bound proof of them can be easily adapted to our variation.

5 Concluding Remarks

For the sorting problem, several algorithms running in $O(n(1 + \log(1 + k/n)))$ time [12, 13, 20, 21, 22, 23] have been presented, and there is a tight lower bound [16]. This lower bound also applies to the first formulation, and there is a gap between this lower bound and the running time of our algorithm. It is desirable to find an optimal algorithm.

For further investigation, we can think of other presortedness measures, and other geometric problems that can be thought of as permutation problems. A lot of questions remain unsolved, and we hope that this is a stimulating line of research.

References

1. Baran, I., Demaine, E.D.: Optimal adaptive algorithms for finding the nearest and farthest point on a parametric black-box curve. Internat. J. Comput. Geom. Appl. 15, 327–350 (2005)
2. Barbay, J., Chen, E.Y.: Convex hull of the union of convex objects in the plane: an adaptive analysis. In: Proc. 20th CCCG, pp. 47–51 (2008)
3. Barbay, J., Golynski, A., Munro, J.I., Rao, S.S.: Adaptive searching in succinctly encoded binary relations and tree-structured documents. Theor. Comput. Sci. 387, 284–297 (2007)
4. Barbay, J., Kenyon: Adaptive intersection and t-threshold problems. In: Proc. 13th SODA, pp. 390–399 (2002)
5. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Comm. ACM 18, 509–517 (1975)
6. Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. J. Comput. Syst. Sci. 7, 448–461 (1972)
7. Bose, P., Maheshwari, A., Morin, P., Morrison, J., Smid, M., Vahrenhold, J.: Space-efficient geometric divide-and-conquer algorithms. Comput. Geom. 37, 209–227 (2007)
8. Brodal, G.S., Fagerberg, R., Moruz, G.: Cache-aware and cache-oblivious adaptive sorting. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 576–588. Springer, Heidelberg (2005)
9. Brönnimann, H., Chan, T.M., Chen, E.Y.: Towards in-place geometric algorithms and data structures. In: Proc. 20th SCG, pp. 239–246 (2004)
10. Brönnimann, H., Iacono, J., Katajainen, J., Morin, P., Morrison, J., Toussaint, G.T.: Space-efficient planar convex hull algorithms. Theor. Comput. Sci. 321, 25–40 (2004)
11. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Adaptive set intersections, unions, and differences. In: Proc. 11th SODA, pp. 743–752 (2000)
12. Elmasry, A.: Priority queues, pairing, and adaptive sorting. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 183–194. Springer, Heidelberg (2002)
13. Elmasry, A., Fredman, M.L.: Adaptive sorting: an information theoretic perspective. Acta Inform. 45, 33–42 (2008)
14. Estivill-Castro, V., Wood, D.: Practical adaptive sorting. In: Dehne, F., Fiala, F., Koczkodaj, W.W. (eds.) ICCI 1991. LNCS, vol. 497, pp. 47–54. Springer, Heidelberg (1991)
15. Estivill-Castro, V., Wood, D.: A survey of adaptive sorting algorithms. ACM Comput. Surveys 24, 441–476 (1992)
16. Guibas, L.J., McCreight, E.M., Plass, M.F., Roberts, J.R.: A new representation of linear lists. In: Proc. 9th STOC, pp. 49–60 (1977)
17. Kapoor, S., Ramanan, P.: Lower bounds for maximal and convex layers problems. Algorithmica 4, 447–459 (1989)

18. Kirkpatrick, D.G., Seidel, R.: The ultimate planar convex hull algorithm? SIAM J. Comput. 15, 287–299 (1986)
19. Levcopoulos, C., Lingas, A., Mitchell, J.S.B.: Adaptive algorithms for constructing convex hulls and triangulations of polygonal chains. In: Penttonen, M., Schmidt, E.M. (eds.) SWAT 2002. LNCS, vol. 2368, pp. 80–89. Springer, Heidelberg (2002)
20. Levcopoulos, C., Petersson, O.: Splitsort—An adaptive sorting algorithm. Infor. Proc. Lett. 39, 205–211 (1991)
21. Levcopoulos, C., Petersson, O.: Adaptive Heapsort. J. Algor. 14, 395–413 (1993)
22. Mannila, H.: Measures of presortedness and optimal sorting algorithms. IEEE Trans. Comput. 34, 318–325 (1985)
23. Mehlhorn, K.: Data Structures and Algorithms. Sorting and Searching, vol. 1. Springer, Heidelberg (1984)
24. Megiddo, N.: Linear programming in linear time when the dimension is fixed. J. ACM 31, 114–127 (1984)
25. Pagh, A., Pagh, R., Thorup, M.: On adaptive integer sorting. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 556–567. Springer, Heidelberg (2004)

New Algorithms for Barrier Coverage with Mobile Sensors

Xuehou Tan[1] and Gangshan Wu[2]

[1] Tokai University, 4-1-1 Kitakaname, Hiratsuka 259-1292, Japan
[2] State Key Lab. for Novel Software Technology, Nanjing University, China
tan@wing.ncc.u-tokai.ac.jp

Abstract. Monitoring and surveillance are important aspects in modern wireless sensor networks. In applications of wireless sensor networks, it often asks for the sensors to quickly move from the interior of a specified region to the region's perimeter, so as to form a barrier coverage of the region. The region is usually given as a simple polygon or even a circle. In comparison with the traditional concept of full area coverage, barrier coverage requires fewer sensors for detecting intruders, and can thus be considered as a good approximation of full area coverage.

In this paper, we present an $O(n^{2.5} \log n)$ time algorithm for moving n sensors to the perimeter of the given circle such that the new positions of sensors form a regular n-gon and the maximum of the distances travelled by mobile sensors is minimized. This greatly improves upon the previous time bound $O(n^{3.5} \log n)$. Also, we describe an $O(n^4)$ time algorithm for moving n sensors, whose initial positions are on the perimeter of the circle, to form a regular n-gon such that the sum of the travelled distances is minimized. This solves an open problem posed in [2]. Moreover, our algorithms are simpler and have more explicit geometric flavor.

1 Introduction

Wireless Sensor Networks (WSN) are envisioned to be developed for a wide range applications. A WSN is composed of a large number of sensor nodes, which are densely deployed either inside the phenomenon or very close to it [1]. Each sensor node is equipped with a sensing device, a low computational capacity processor, a short-range wireless transmitter-receiver and a limited battery-supplied energy. Sensor nodes monitor some surrounding environmental phenomenon, process the data obtained and forward these data towards a base station. These characteristics of a WSN require that sensor network protocols and algorithms possess self-organizing capabilities, i.e., sensors are able to cooperate in order to organize and perform networking tasks efficiently.

A typical application of WSN is to monitor a specified region either for measuring purposes or for reporting various types of activities (e.g., fire alarms, calamities, etc). Another application concerns security and safety systems, such as, detecting intruders (or movement thereof) around infrastructure facilities and regions. Particularly, it often asks to monitor an area so as to detect intruders as they penetrate the protected area or as they cross the area border.

D.T. Lee, D.Z. Chen, and S. Ying (Eds.): FAW 2010, LNCS 6213, pp. 327–338, 2010.

For example, research efforts are currently under way to extend the scalability of wireless sensor networks so that they can be used to monitor international border as well [8,11].

The study of *barrier coverage* with mobile sensors was originated in [3,11], and later in [2]. Differing from the traditional concept of *full coverage*, it asks to cover the entire deployment region by guaranteeing that there is no path through this region that can be traversed undetectedly by an intruder, i.e., all crossing paths through the region are covered by sensors [2,3,11]. Since mobile sensors are allowed to move inside the deployment region, a crossing path may occur occasionally. So, an interesting problem is to reposition the sensors quickly so as to repair the existing security hole and thereby detect intruders [2]. Since barrier coverage requires fewer sensors for detecting intruders, it thus gives a good approximation of full area coverage. The planar region on which sensors move is usually represented by a simple polygon or even a circle.

In this paper, we study the problem of moving n sensors to the perimeter of a circular region to form a regualr n-gon such that either the maximum of the distances travelled by mobile sensors or the sum of the travelled distances is minimized. An efficient solution to the *min-max* or *min-sum* problem is important, as the energy required by a mobile sensor is an increasing function of the distance it travels. First, we present an $O(n^{2.5} \log n)$ time algorithm for moving n sensors to the perimeter of the given circle such that the new positions of sensors form a regular n-gon and the maximum of the distances travelled by mobile sensors is minimized. This improves upon the previous time bound $O(n^{3.5} \log n)$ [2]. Also, we describe an $O(n^4)$ time algorithm for moving n sensors, whose initial positions are on the perimeter of the given circle, to form a regular n-gon such that the sum of the travelled distances is minimized. This solves an open problem posed in Section 5.5 of [2]. Moreover, since our algorithms are based on some properties from elementary geometry, they are simple and easy to implement.

2 Problem Definition and Previous Work

Suppose that mobile sensors are working inside a planar region. As discussed in [2,3], individual sensors are able to locally determine the existence of barrier coverage, even when the region is arbitrarily shaped. For simplicity, the region is usually delimited by a simple polygon or even by a circle. In this paper, we mainly focus on the problem of moving n sensors to the perimeter of a unit-radius circular region such that the new positions of sensors form a regular n-gon and thereby give barrier coverage, assuming that the mobile sensors have detected the existence of a crossing path. We assume that the sensors are location aware (i.e., they know their geometric coordinates) and know the center of the given circle. Assume also that the range of the sensor's transmitter-receiver is always longer than an edge of the regular n-gon; otherwise, barrier coverage is impossible.

Denote by C the given circular region. For n sensors, denote by A_1, A_2, ..., A_n their initial positions in the interior or on the perimeter of C, and A'_1, A'_2, ..., A'_n their goal positions on the perimeter of C, which form a regular n-gon.

Moreover, denote by AA' the line segment with two endpoints A and A', and $|AA'|$ the length of AA' (i.e., the Euclidean distance between A and A'). Then, one can easily define the following two problems:

1. **The min-max problem**: Minimizing the maximum of the distances travelled by the sensors, i.e., $min\ \{max_{i=1}^{n}|A_iA_i'|\}$.
2. **The min-sum problem**: Minimizing the sum of the distances travelled by the sensors, i.e., $min\ \sum_{i=1}^{n}|A_iA_i'|$.

An $O(n^{3.5}\log n)$ time algorithm has been proposed by Bhattacharya et al. to solve the min-max problem [2]. For a simple polygon with m vertices, an $O(mn^{3.5}\log n)$ time solution is also given. Their algorithms are based on the parametric searching technique, which requires many sophisticated procedures (e.g., a sorting algorithm with an unknown optimal value [4], and a parallel sorting network [12]). As many other geometric applications of parametric searching, the necessary condition for giving an optimal solution is not explictly described [2]. These drawbacks of parametric searching have previously been pointed out in the literature (see Section 1 of [9]).

Two approximation algorithms for the min-sum problem for circular regions are further provided in [2]. A simple $O(n^2)$ time solution to a special version of the min-sum problem, in which both the initial positions and the movements of all sensors are limited on the perimeter of the given circle, is also given. Again, the similar result was extended to polygonal regions [2]. Whether a polynomial-time solution to the min-sum problem exists is left open, even when the initial positions of all sensors are on the perimeter of the given circle (see Section 5.5 of [2]).

3 Min-max Problem

Suppose that C is a unit-radius circle, and o is the center of C. Denote by ∂C the perimeter of the circle C. Moreover, denote by λ_C the optimal solution to the min-max problem for the circle C, i.e., $\lambda_C = min\ \{max_{i=1}^{n}|A_iA_i'|\}$. Clearly, $\lambda_C \leq 2$.

In the following, we study the geometric properties of the points on ∂C that may contribute to the optimum λ_C, and then present our new algorithm for computing λ_C.

3.1 Geometric Properties of the Boundary Points Related to λ_C

Denote by X_i the point of ∂C, which is closest to A_i. Clearly, X_i is an intersection point of ∂C with the line passing through A_i and o. Denote by Y_i the other intersection point, which is the point of ∂C furthest from A_i. See Figure 1(a).

The following properties are important to our algorithm for the min-max probem.

Lemma 1. *Suppose that an optimal solution to the min-max problem is obtained with $\lambda_C = |A_iA_i'|$, for some i ($1 \leq i \leq n$). Then, either A_i' is the point X_i, or there exists another sensor A_j ($j \neq i$) such that $\lambda_C = |A_jA_j'|$ also holds.*

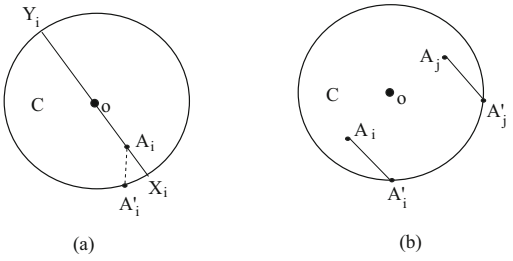


Fig. 1. (a) The points X_i and Y_i on ∂C; (b) $|A_iA_i'| = |A_jA_j'|$

In the latter case, a slight rotation of the regular n-gon giving λ_C in either direction monotonically increases one of the two distances $|A_iA_i'|$ and $|A_jA_j'|$, and decreases the other.

Proof. Assume first that in the obtained optimal solution, the sensor A_i is the only one satisfying $\lambda_C = |A_iA_i'|$, but A_i' is not the point X_i. So $|A_iX_i| < |A_iA_i'|$ holds (Figure 1(a)). Let us rotate the regular n-gon giving λ_C by moving the vertex A_i' towards X_i, with a very small distance ϵ. Clearly, the distance function between A_i and A_i' decreases monotonically during the rotation of the n-gon. Denote by A_1'', A_2'', ..., A_n'' the new positions of the sensors after the rotation stops. Since ϵ is arbitrarily small and A_i is the only one satisfying $\lambda_C = |A_iA_i'|$, we have $|A_iA_i''| \geq |A_kA_k''|$ for all $k \neq i$, and moreover, $|A_iA_i''| < |A_iA_i'|$ holds; it contradicts that λ_C $(= |A_iA_i'|)$ gives the optimal solution to the min-max problem.

Suppose now that there exists another sensor A_j such that $\lambda_C = |A_jA_j'|$ $(j \neq i)$ also holds (Figure 1(b)). A slight rotation of the regular n-gon giving λ_C in either direction cannot make both $|A_iA_i''| < |A_iA_i'|$ and $|A_jA_j''| < |A_jA_j'|$ hold, where A_i'' and A_j'' denote the new positions of A_i' and A_j' after the rotation stops; otherwise, it contradicts the equations $\lambda_C = |A_iA_i'|$ and $\lambda_C = |A_jA_j'|$. This implies that the rotation of the regular n-gon giving λ_C increases one of the two distances $|A_iA_i'|$ and $|A_jA_j'|$, but decreases the other. The proof is complete. □

The points of ∂C satisfying the conditions described in Lemma 1 clearly contribute to the candidate values for λ_C. The points X_h of all sensors A_h $(1 \leq h \leq n)$ can simply be found. So an important task is to find all the pairs (A_i, A_j) $(i \neq j)$ such that the distance from A_i to a vertex of a regular n-gon equals to the distance from A_j to another vertex of the n-gon, and a slight rotation of the n-gon in either direction monotonically increases one of the two distances but decreases the other. For simplicity, we call such distances the *equal distances*.

Let P be an arbitrary regular n-gon with the vertices P_1, P_2, ..., P_n on ∂C. In order to find all equal distances, we simulate below a clockwise rotation of the polygon P on ∂C with an arc distance $2\pi/n$. First, compute the distances

between all pairs of a sensor and a vertex of P, and sort these n^2 distances into the sequence, say, $d_1 < d_2 < \ldots < d_{n^2}$. Denote by D the resulting sequence of these distances. For any two adjacent distances in D, we then check whether they can produce an equal distance during the procedure of rotating P. Note that the two sensors related to that equal distance as well as the two intervals of ∂C (of arc length $2\pi/n$) on which the two vertices of P move are known. Moreover, since one of the two distances increases but the other decreases to the equal distance, the exact distance between the goal positions of two sensors is also known. (The distance between two goal positions can be represented as $2sin(w\pi/n)$, where $w(\leq n)$ is a known, positive integer.) Thus, we can determine in constant time whether the equal distance for a pair of adjacent distances in D can be attained. If *yes*, report the found equal distance. From the initial sequence $d_1, d_2, \ldots, d_{n^2}$, we can thus obtain some equal distances. Denote by E the set of the found equal distances.

To achieve an equal distance e, the polygon P is rotated on ∂C with an arc distance from its *initial* position. Denote by $r(e)$ $(\leq 2\pi/n)$ this arc distance (from the initial position of P) required to obtain the equal distance e. Let R be the set of these arc distances $r(e)$, and r_{min} the smallest arc distance in R.

The clockwise rotation of P with the arc distance $2\pi/n$ can then be performed as follows. First, take out (i.e., delete) r_{min} from the set R. Since an equal distance happens only between two adjacent distances in D, the first equal distance occurs when the polygon P is rotated on ∂C with the arc distance r_{min}. Afterwards, the two adjacent distances, which contributed to r_{min}, have to be exchanged in the sequence D. The change of these two distances in D may introduce at most two new equal distances, and thus, we further check whether these two equal distances can be attained. If a new equal distance is ever found, we compute the corresponding arc distance from the initial position of P, and then insert it into R. Next, take out the current arc distance r_{min} from R and continue to rotate P on ∂C according to (the remaining value of) r_{min}. In this way, the polygon P is gradually rotated, and whenever the arc distance represented by r_{min} is reached, we maintain the sequence D of n^2 distances between sensors and the current vertices of P, and the set R of the arc distances, which are computed from the found equal distances. All equal distances are clearly reported after the rotation of P is complete.

Lemma 2. *Let m be the number of the equal distances. Then, $m = O(n^3)$.*

Proof. Let $A_i(P_x)$ denote the distance function from a sensor A_i to a vertex P_x of the n-gon P. Clearly, the function $A_i(P_x)$ increases or decrease monotonically in the procedure of rotating P, except that the interval of ∂C on which P_x moves contains the point X_i or Y_i; in this case, we divide that interval of ∂C into two sub-intervals such that $A_i(P_x)$ is monotone in either sub-interval. All functions $A_i(P_x)$, $P_x \in P$, can thus be grouped into two sets S_{i1} and S_{i2} such that the functions in the set S_{i1} monotonically increase and the functions in S_{i2} monotonically decrease. The union of the intervals of ∂C, on which the vertices

P_x of all functions of S_{i1} (resp. S_{i2}) move, is the clockwise chain of ∂C from X_i to Y_i, or from Y_i to X_i. Hence, all functions in S_{i1}, or in S_{i2} can be considered as a single distance function from the sensor A_i, which is also monotone. Similarly, we can define the monotone functions $A_j(P_y)$, for all other sensors A_j and all polygon vertices P_y.

Let us now give an upper bound on the number of the equal distances. In the procedure of rotating P with the arc distance $2\pi/n$, any decreasing (resp. increasing) function $A_j(P_y)$ can produce at most one equal distance with a function of S_{i1} (resp. S_{i2}), $i \neq j$. This is because the function $A_j(P_y)$ is monotone, and all functions in S_{i1} (resp. S_{i2}) can be considered as a monotone distance function from A_i. Since both the number of the sensors A_j $(j \neq i)$ and the number of the vertices P_y are no more than n, the sensor A_i can contribute to $O(n^2)$ equal distances. Therefore, we have $m = O(n^3)$. □

3.2 Algorithm

Let p be a point on ∂C, and d the distance between p and an arbitrary sensor. We can then give an algorithm for determining whether $\lambda_C \leq d$ [2].

Algorithm Distance-Test

1. Compute the regular n-gon by fixing one of its vertices at p, and denote by $B_1, B_2, \ldots, B_n$ the vertices of the obtained n-gon.
2. Construct a bipartite graph H_d between the sensors $A_1, A_2, \ldots, A_n$ and the vertices $B_1, B_2, \ldots, B_n$; sensor A_i is linked to vertex B_j $(1 \leq i, j \leq n)$ if and only if $|A_iB_j| \leq d$.
3. Check whether there exists a perfect matching in H_d. If *yes*, report $\lambda_C \leq d$; otherwise, report $\lambda_C > d$.

The time complexity of the algorithm **Distance-Test** is $O(n^{2.5})$. This is because the first two steps of **Distance-Test** clearly take $O(n^2)$ time, and the last step requires $O(n^{2.5})$ time to check whether there exists a perfect matching in the bipartite graph H_d [6].

To give a simple solution to the min-max problem, one can run a binary search over all the distances $|A_iX_i|$ $(1 \leq i \leq n)$ and the equal distances e_j $(1 \leq j \leq m)$ using the fixed-size decision algorithm **Distance-Test** to determine whether the optimum λ_C is larger than, smaller than or equal to the selected distance d. Clearly, the smallest of the values d satisfying $\lambda_C \leq d$ gives the answer to the min-max problem.

Let us now describe a more efficient algorithm for the min-max problem. Again, denote by P a regular n-gon with the vertices $P_1, P_2, \ldots, P_n$ on ∂C. First, we perform a binary search over all the distances between sensors and the initial positions of vertices of P, i.e., $d_1, d_2, \ldots, d_{n^2}$, to find two distances d_k, d_{k+1} such that $d_k < \lambda_C \leq d_{k+1}$. Without loss of generality, we assume that

$d_0 = 0$, $d_{n^2+1} = 2$, and $0 \leq k \leq n^2$. In the following, we show that there are at most $O(n^2)$ equal distances e, with $d_k < e \leq d_{k+1}$.

Let P^i $(1 \leq i \leq n)$ denote the regular n-gon obtained by fixing one of its vertices at the point X_i. Suppose that the vertices of P^i are indexed clockwise, starting from the vertex $P^i_1 = X_i$. Clearly, there are $\lceil (n-2)/2 \rceil$ pairs of the vertices in P^i, whose distances to the sensor A_i are the same. Let $d^i_1 < d^i_2 < \ldots < d^i_{\lceil n/2 \rceil + 1}$ denote the sequence of the distances from A_i to all vertices of P^i (including P^i_1) and the point Y_i as well when n is odd. When n is odd, the point Y_i is not the vertex of P^i. But, as discussed above, some distance function needs to be divided into two monotone sub-functions using the point Y_i.

Lemma 3. *Let P be an arbitrary regular n-gon on ∂C, and d_1, d_2, …, d_{n^2} the increasing order of the distances between all sensors and the vertices of P. Assume that $d_0 = 0$ and $d_{n^2+1} = 2$, and that we know $d_k < \lambda_C \leq d_{k+1}$ for some k, $0 \leq k \leq n^2$. Let $m(k)$ be the number of the equal distances e, with $d_k < e \leq d_{k+1}$. Then, $m(k) = O(n^2)$, and all of these equal distances can be computed in $O(n^2 \log n)$ time.*

Proof. Suppose that we have known $d_k < \lambda_C \leq d_{k+1}$. For any polygon P^i $(1 \leq i \leq n)$, the range $(d_k, d_{k+1}]$ is clearly contained in $[d^i_j, d^i_{j+2}]$ for some $j < \lceil n/2 \rceil$, or in $[d^i_j, d^i_{j+1}]$ when $j = \lceil n/2 \rceil$. Also, denote by $A_i(P^i_x)$ the distance function from the sensor A_i to a vertex P^i_x of the n-gon P^i. Since the distance functions from A_i to all vertices of P^i can be grouped into two monotone functions in the rotation of P^i with the arc distance $2\pi/n$, at most four distance functions $A_i(P^i_x)$ may vary in the range $(d_k, d_{k+1}]$. These distance functions from A_i can be found in $O(\log n)$ time, provided that the increasing order of the distances d^i_1, d^i_2, …, $d^i_{\lceil n/2 \rceil + 1}$ is known.

Consider now the procedure of rotating the polygon P on ∂C clockwise with the arc distance $2\pi/n$. Suppose that all the polygons P^1, P^2, …, P^n are also rotated with the arc distance $2\pi/n$ simultaneously. Assume that all the $O(n)$ distance functions, which vary in the range $(d_k, d_{k+1}]$, have been sorted according to their initial values (i.e., using the corresponding distances d^i_j, $1 \leq i, j \leq n$). Denote by $D(k)$ the increasing order of these distance functions. For any two adjacent distances in $D(k)$, we then check whether they can be equal in the procedure of rotating the polygon P. If *yes*, report that equal distance. Also, we can simply check whether the found equal distance is between d_k and d_{k+1}. From the initial sequence $D(k)$, we can compute a set of equal distances. Denote by $R(k)$ the set of the arc distances, which are required to attain the found equal distances. As in the proof of Lemma 2, the polygon P can then be rotated according to the arc set $R(k)$. When the set $R(k)$ becomes empty in the procedure of rotating P, we obtain all the equal distances e, with $d_k < e \leq d_{k+1}$. The number $m(k)$ of the equal distances e, with $d_k < e \leq d_{k+1}$, is $O(n^2)$. This is because $D(k)$ contains $O(n)$ monotone functions in the whole procedure of rotating P and a pair of adjacent distances in $D(k)$ at any instant time of the rotation can contribute to at most one equal distance.

Finally, consider the time required to compute these equal distances. First, the sequences of the distances d_1^i, d_2^i, ..., $d_{\lceil n/2 \rceil+1}^i$, for all i $(1 \leq i \leq n)$, can totally be computed in time $O(n^2 \log n)$. The initial sequence $D(k)$ of the distance functions, which vary in the range $(d_k, d_{k+1}]$, can then be found in $O(n \log n)$ time. When an equal distance occurs between two adjacent distances of $D(k)$ in the procedure of rotating P, we insert its corresponding arc distance into the set $R(k)$. For efficiency, $R(k)$ is maintained in a priority queue. Hence, an insertion of an arc distance to $R(k)$ as well as a deletion of the smallest arc distance from $R(k)$ takes $O(\log n)$ time. So the total time required is $O(n^2 \log n)$. □

By now, we can give our algorithm for the min-max problem. First, perform a binary search over the distances d_0, d_1, ..., d_{n^2+1} using the fixed-size decision algorithm **Distance-Test** to determine whether the optimum λ_C is larger than, smaller than or equal to the selected distance. After this binary search is done, we can assume that $d_k < \lambda_C \leq d_{k+1}$ for some k, $0 \leq k \leq n^2$. Next, rotate the polygon P with the arc length $2\pi/n$ to find all the equal distances e, with $d_k < e \leq d_{k+1}$. Furthermore, we sort these $m(k)$ equal distances and the other n distances $|A_j X_j|$ $(1 \leq j \leq n)$. Denote the resulting sequence by $ed_1 < ed_2 < \ldots < ed_{n+m(k)}$. Since the optimum λ_C is one of these $n + m(k)$ values (Lemmas 1 and 3), it suffices to run another binary search using the algorithm **Distance-Test**. We conclude the algorithm in the following:

Algorithm Min-Max

1. Let P be an arbitrary regular n-gon on ∂C, and d_1, d_2, ..., d_{n^2} the increasing order of the distances between all sensors and the vertices of P. Assume also that $d_0 = 0$ and $d_{n^2+1} = 2$.
2. Run a binary search over the distances d_0, d_1, ..., d_{n^2+1} using the fixed-size decision algorithm **Distance-Test** to find two values d_k, d_{k+1} $(0 \leq k \leq n^2)$ such that $d_k < \lambda_C \leq d_{k+1}$. (The regular n-gon used in **Distance-Test** is constructed according to the selected distance.)
3. For every i $(1 \leq i \leq n)$, place the regular n-gon P^i on ∂C by fixing one of its vertices at the point X_i, and then sort the distances from A_i to all vertices of P^i and the point Y_i as well when n is odd, into the sequence $d_1^i < d_2^i < \ldots < d_{\lceil n/2 \rceil+1}^i$. Next, find at most $4n$ vertices of P^i such that their distance functions (from A_i) vary in the range $(d_k, d_{k+1}]$ when P^i is rotated with the arc distance $2\pi/n$.
4. Rotate the polygon P on ∂C with the arc distance $2\pi/n$ to compute all the equal distances e, with $d_k < e \leq d_{k+1}$. Then, sort these $m(k)$ equal distances and the other n distances $|A_j X_j|$ $(1 \leq j \leq n)$ into the sequence , say, $ed_1 < ed_2 < \ldots < ed_{n+m(k)}$.
5. Run a binary search over the distances ed_1, ed_2, ..., $ed_{n+m(k)}$ using the fixed-size decision algorithm **Distance-Test** to determine whether λ_C is larger than, smaller than or equal to the selected distance.
6. Report the smallest of the values ed_j satisfying $\lambda_C \leq ed_j$.

It follows from the discussion made above that Steps 2 and 5 of **Min-Max** take $O(n^{2.5} \log n)$ time, which clearly dominates the time complexity of the algorithm **Min-Max**. Hence, we obtain the main result of this paper:

Theorem 1. *The min-max problem for a given circle can be solved in* $O(n^{2.5} \log n)$ *time.*

4 Min-sum Problem: When Sensors Are Initially on the Perimeter of the Circle

This section presents an $O(n^4)$ time algorithm for a special version of the min-sum problem, in which all sensors are initially on the perimeter of the unit-radius circle C. Our result solves an open problem posed in [2].

Let A_1, A_2, ..., A_n denote the initial positions of n sensors on ∂C, and A'_1, A'_2, ..., A'_n their goal positions on ∂C. Denote by Δ_C the optimal solution to the min-sum problem for C, i.e., $\Delta_C = min \sum_{i=1}^{n} |A_i A'_i|$. The following property is important to the solution of our min-sum problem.

Lemma 4. *Suppose that all sensors are initially on the perimeter of the unit-radius circle* C*. In any assignment between the initial and goal positions of* n *sensors, which gives* Δ_C*, there exists some sensor* A_x *(*$1 \leq x \leq n$*) such that* $A_x = A'_x$*.*

Proof. The proof is by contradiction. Assume that in any assignment between the initial and goal positions of n sensors, which gives Δ_C, all sensors move, i.e., $A_i \neq A'_i$ for all i, $1 \leq i \leq n$. For our purpose, we construct below a line segment L such that the length L is equal to Δ_C. To avoid confusion, we use the small letters 'a_k' ('a'_k') to represent the points A_k (A'_k) on L. See Figure 2(b). First, fix a segment $A_s A'_s$ for an aribitrary index s, in an assignment giving Δ_C. Then, draw all other segments contributed to Δ_C, in an arbitrary order, on the extension of the segment $A_s A'_s$ by connecting the point A'_y of the segment $A_y A'_y$ to the previously existed point A_x. Assume that a_t is the last endpoint obtained in constructing the segment L. Since a'_s is the other endpoint of L, we have $|a_t a'_s| = \Delta_C$. See Figure 2(b) for an example, where $a'_s = a'_3$ and $a_t = a_1$.

Let us now move all points A'_j on ∂C clockwise, by a very small arc distance α, say, $n\alpha < \pi/4$. Assume that no point A_j is passed over during this clockwise rotation of the n-gon (it is always possible since α can be arbitrarily small). Denote by B_j the new position of A'_j ($1 \leq j \leq n$), and Δ_1 the solution of the min-sum problem in which every sensor A_j moves to B_j. See Figure 2(a). Since three points A_j, A'_j and B_j ($1 \leq j \leq n$) are on ∂C, the angle $\angle A'_j A_j B_j$ is α. Next, we translate the segment $A_t B_t$ such that the translated segment, denoted by $a_t b_t$, and the segment $a_t a'_t$ (on L) form the angle α at the point a_t. See Figure 2(b). Draw all other segments in the assigment giving Δ_1 on the extension of the segment $a_t b_t$, in the reversed order of the segments added to construct L. Denote by L_1 the resulting segment. See Figure 2(b). Clearly, b_s is the other endpoint of L_1. Also, we have $|a_t b_s| = \Delta_1$.

Analogously, we move all points A'_j on ∂C counterclockwise, by the same arc distance α. Assume also that no point A_j is passed over during this counterclockwise rotation of the n-gon. Denote by E_j the new position of A'_j $(1 \leq j \leq n)$, and Δ_2 the solution of the min-sum problem in which every sensor A_j moves to E_j. Similarly, we can obtain a line segment L_2, with two endpoints a_t and e_s, of length Δ_2. See Figure 2(b). Without loss of generality, assume that L_1 and L_2 are to the different sides of L. Thus, L_1 and L_2 form the angle 2α at the point a_t.

In the above construction of L and L_1 (resp. L_2), only the translation and the rotation of segments are empolyed. Hence, two points a'_s and b_s (resp. a'_s and e_s) can be connected by n arcs of length α, which are scanned by all sensors in transforming the assignment for Δ_C into that for Δ_1 (resp. Δ_2). These n arcs can thus be replaced by a long arc of length $n\alpha$, i.e., a'_s and b_s (resp. a'_s and e_s) are connected by an arc of length $n\alpha$ $(< \pi/4)$. It immediately implies that three points a'_s, b_s and e_s are passed by another unit-radius circle, say, C'. See Figure 2(b). Since the angle formed by L_1 and L_2 (i.e., two segments a_tb_s and a_te_s) at a_t is 2α, the point a_t cannot be contained in the interior of the circle C' (otherwise, since the arc distance between b_s and e_s is $2n\alpha$, we would have $2\alpha > 2n\alpha$ $(n \geq 2)$, a contradiction). The segment L then has two common points with $\partial C'$. If the center of C' happens to be on L, we have $\Delta_1 < \Delta_C$ and $\Delta_2 < \Delta_C$, contradicting that Δ_C gives an optimal solution to the min-sum problem. In the case that the center of C' is not on L, either $\Delta_1 < \Delta_C$ or $\Delta_2 < \Delta_C$ holds, a contradiction again. This completes the proof. □

Based on Lemma 4, the min-sum problem for the circle C can be solved by fixing a vertex of the regular n-gon at the initial position of every sensor once, then solving the weighted matching problem (n times) in a complete bipartite graph such that one subset of its nodes represents the initial positions of all sensors and the other represents the goal positions of sensors, and finally reporting the smallest of the obtained solutions to all weighted matching instances.

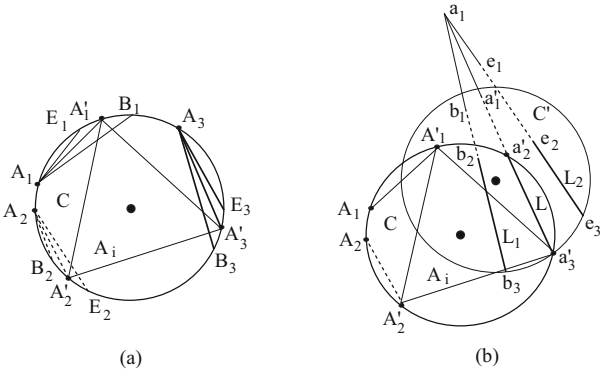


Fig. 2. Illustration of the proof of Lemma 4

Algorithm Min-Sum

1. For each $i = 1, 2, \ldots, n$, do the following:
   (a) Place a regular n-gon P on ∂C by fixing one of its vertices at the sensor A_i. Then, construct a complete bipartite graph H_i between the sensors $A_1, A_2, \ldots, A_n$ and the vertices $P_1, P_2, \ldots, P_n$ of the n-gon (i.e., any of n sensors is linked to all polygon vertices). Moreover, define the Euclidean distance between a sensor and a vertex of P as the weight of the edge of H_i connecting the corresponding nodes.
   (b) Compute an optimal solution $\Delta(H_i)$ to the weighted bipartite matching problem for the graph H_i (i.e., finding the optimal assignment of the n sensors to the vertices of P in the weighted bipartite graph H_i).
2. Report the smallest of all found solutions $\Delta(H_i)$.

Since the number of nodes of the graph H_i is $2n$, the Hungarian method can solve in $O(n^3)$ time the weighted matching problem in the complete bipartite graph H_i [10]. Hence, we have the following result.

Theorem 2. *Suppose that all sensors are initially located on the perimeter of the given circle. Then, the min-sum problem can be solved in $O(n^4)$ time.*

5 Concluding Remarks

In this paper, we have presented an $O(n^{2.5} \log n)$ time algorithm for moving n sensors to the perimeter of the given circle such that the new positions of sensors form a regular n-gon and the maximum of the distances travelled by mobile sensors is minimized. This greatly improves upon the previous time bound $O(n^{3.5} \log n)$. Also, we have described an $O(n^4)$ time algorithm for moving n sensors, given on the perimeter of the circle, to form a regular n-gon such that the sum of the travelled distances is minimized. This solves an open problem posed in [2]. Moreover, our algorithms are simple and easy to implement.

There are several open questions which are interesting for further research. First, whether the min-sum problem is NP-hard is not known. The answer might be negative, since it seems quite difficult to specify a finite number of candidate points on the perimeter of the given circle such that an optimal solution can be computed from these candidate points. Also, it is interesting to extend our methods to polygonal or convex regions. For a polygonal or convex polygon, the sensors are required to move to the perimeter of the polygon such that the polygonal distance along the perimeter between any two consecutive sensors are the same [2]. Finally, although our algorithms as well as the previous algorithms given in [2] were developed for a wireless sensor network, we have assumed a centralized control server, where nodes are connected using a gateway. The distributed self-deployment algorithms for full/barrier coverage, which consider various designing strategies, such as oblivious robots, uniformity and system lifetime, have been studied in [5,7]. Whether the distributed version of our algorithms can be developed is an interesting question for further work.

References

1. Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless Sensor Networks: A Survey. Computer Networks (Elsevier) Journal, 393–422 (March 2002)
2. Bhattacharya, B., Burmester, M., Hu, Y., Kranakis, E., Shi, Q., Wiese, A.: Optimal movement of mobile sensors for barrier coverage of a planar region. Theoretical Computer Science 410, 5515–5528 (2009)
3. Chen, A., Kumar, S., Lai, T.H.: Designing localized algorithms for barrier coverage. In: Proc. 13th ACM Int. Conf. on Mobile Computing and Networking, pp. 63–73 (2007)
4. Cole, R.: Slowing down sorting networks to obtain faster algorithms. Journal of the ACM 34(1), 200–208 (1987)
5. Défago, X., Souissi, S.: Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. Theoretic Computer Science 396, 97–112 (2008)
6. Hopcroft, J.E., Karp, R.M.: An $n^{2.5}$ algorithm for maximum matching in bipartite graphs. SIAM J. Comput. 2(4), 225–231 (1973)
7. Heo, N., Varshney, P.K.: Energy-efficient deployment of intelligent mobile sensor networks. IEEE Trans. Systems, Man and Cybernetics, Part A 35(1), 78–92 (2005)
8. Hu, S.S.: 'Virtual Fence' along border to be delayed, Washington Post (February 28, 2008)
9. Katz, M.J., Sharir, M.: An expander-based approach to geometric optimization. In: Proc. 10th ACM Symp. on Computational Geometry, pp. 198–207 (1993)
10. Kuhn, H.W.: The Hungarian method for the assignment problem. Naval Research Logistics Quarterly 2, 83–97 (1955)
11. Kumar, S., Lai, T.H., Arora, A.: Barrier coverage with wireless sensors. Wireless Networks 13(6), 817–834 (2007)
12. Megiddo, N.: Applying parallel computation algorithms in design of serial algorithms. Journal of the ACM 30(4), 852–865 (1983)

Author Index