Michael Fellows
Xuehou Tan
Binhai Zhu (Eds.)

Frontiers in Algorithmics *and* Algorithmic Aspects in Information and Management

LNCS 7924

Third Joint International Conference, FAW-AAIM 2013
Dalian, China, June 2013
Proceedings

Springer

Lecture Notes in Computer Science 7924

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

Michael Fellows Xuehou Tan Binhai Zhu (Eds.)

Frontiers in Algorithmics *and* Algorithmic Aspects in Information and Management

Third Joint International Conference, FAW-AAIM 2013
Dalian, China, June 26-28, 2013
Proceedings

Springer

Volume Editors

Michael Fellows
Charles Darwin University
School of Engineering and Information Technology
Darwin, NT 0909, Australia
E-mail: michael.fellows@cdu.edu.au

Xuehou Tan
Tokai University
School of High-Technology for Human Welfare
317 Nishimo, Numazu 410-0395, Japan
E-mail: tan@wing.ncc.u-tokai.ac.jp

Binhai Zhu
Montana State University
Department of Computer Science
Bozeman, MT 59717, USA
E-mail: bhz@cs.montana.edu

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-38755-5 e-ISBN 978-3-642-38756-2
DOI 10.1007/978-3-642-38756-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013939182

CR Subject Classification (1998): F.2, G.2, I.3.5, E.1, F.1, J.1, I.2

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

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

Printed on acid-free paper

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

Preface

The papers in this volume were presented at the 7th International Frontiers in Algorithmics Workshop and the 9th International Conference on Algorithmic Aspects in Information and Management (FAW-AAIM 2013), jointly held during June 26-28, 2013, in Dalian, China. The topics cover most areas in algorithms, combinatorial optimization and their applications.

Submissions to the conference this year were conducted electronically. A total of 60 papers were submitted, of which 33 were accepted. The papers were evaluated by an international Program Committee overseen by the Program Committee Co-chairs: Michael Fellows, Xuehou Tan, and Binhai Zhu. The Program Committee consisted of Heep-Kap Ahn, Rong Chen, Ovidiu Daescu, Gregory Gutin, Xin, Han Tomio Hirata, Wing-Kai Hon, Hiro Ito, Kazuo Iwama, Naoki Katoh, Tian Liu, Jun Luo, Joseph S.B. Mitchell, Brendan Mumey, Hirotaka Ono, Frances Rosamond, Toshinori Sakai, Jens Stoye, Wing-King Sung, Hing-Fung Ting, Jorge Urrutia, Carola Wenk, Ge Xia, Mingyu Xiao, Yinfeng Xu, Boting Yang, Liping Yuan, Weishi Zhang, and Daming Zhu. It is expected that most of the accepted papers will appear in a more complete form in scientific journals.

The submitted papers were from 20 countries/regions: Australia, Brazil, Canada, China, France, Germany, Hong Kong, India, Israel, Japan, Korea, Mexico, The Netherlands, Pakistan, Poland, Romania, Singapore, Taiwan, UK, and USA. On average, each paper was evaluated by three Program Committee members, assisted in some cases by sub-reviewers. In addition to the selected papers, the conference also included two invited presentations by Jin Akiyama and Daniel Marx.

We thank all the people who made this meeting possible: the authors for submitting papers, the Program Committee members and external reviewers (listed in the proceedings) for their excellent work, and the two invited speakers. Finally, we thank NSF of China, Dalian Maritime University, and SIAM for their support and the local organizers and colleagues for their assistance.

June 2013

Michael Fellows
Xuehou Tan
Binhai Zhu

Organization

Conference Chairs

John Hopcroft	Cornell University, USA
Zuwen Wang	Dalian Maritime University, China

Program Committee Co-chairs

Michael Fellows	Charles Darwin University, Australia
Xuehou Tan	Tokai University, Japan
Binhai Zhu	Montana State University, USA

Program Committee

Heep-Kap Ahn	Pohang University of Science and Technology, Korea
Rong Chen	Dalian Maritime University, China
Ovidiu Daescu	University of Texas at Dallas, USA
Gregory Gutin	University of London, UK
Xin Han	Dalian University of Technology, China
Tomio Hirata	Nagoya University, Japan
Wing-Kai Hon	National Tsing Hua University, Taiwan
Hiro Ito	The University of Electro-Communication, Japan
Kazuo Iwama	Kyoto University, Japan
Naoki Katoh	Kyoto University, Japan
Tian Liu	Beijing University, China
Jun Luo	Shenzhen Institutes of Advanced Technology, China
Joseph S.B. Mitchell	Stony Brook University, USA
Brendan Mumey	Montana State University, USA
Hirotaka Ono	Kyushu University, Japan
Frances Rosamond	Charles Darwin University, Australia
Toshinori Sakai	Tokai University, Japan
Jens Stoye	Bielefeld University, Germany
Wing-King Sung	National University of Singapore, Singapore
Hing-Fung Ting	University of Hong Kong, Hong Kong
Jorge Urrutia	Universidad Nacional Autonoma de Mexico, Mexico
Carola Wenk	Tulane University, USA

Ge Xia	Lafayette College, USA
Mingyu Xiao	University of Electronic Science and Technology, China
Yinfeng Xu	Sichuan University, China
Boting Yang	University of Regina, Canada
Liping Yuan	Hebei Normal University, China
Weishi Zhang	Dalian Maritime University, China
Daming Zhu	Shandong University, China

Organizing Committee

Bo Jiang (Co-chair)	Dalian Maritime University, China
Ansheng Deng (Co-chair)	Dalian Maritime University, China
Xianping Fu	Dalian Maritime University, China
Huawei Zhai	Dalian Maritime University, China
Lijuan Wang	Dalian Maritime University, China
Qi Wei	Dalian Maritime University, China

Additional Reviewers

Tat Wing Chim	Rod Downey	Yuya Higashikawa
Toshimasa Ishii	Mark Jones	Iyad Kanj
Jun Kawahara	Sang-Sub Kim	Yusuke Kobayashi
Neal Koblitz	Yiannis Koutis	Michael Lampis
Gabriele Muciaccia	Yoshio Okamoto	Kazuhisa Seto
Zhengxiang Shen	Wanbin Son	Manuel Sorge
Jessica Sherette	Junichi Teruyama	Akihiro Uejima
Prudence Wong	Tsz Hon Yuen	Chihao Zhang
Yong Zhang		

Table of Contents

Invited Lectures

Contributed Papers

The Square Root Phenomenon in Planar Graphs

Dániel Marx*

Computer and Automation Research Institute,
Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary
dmarx@cs.bme.hu

Most of the classical NP-hard problems remain NP-hard when restricted to planar graphs, and only exponential-time algorithms are known for the exact solution of these planar problems. However, in many cases, the exponential-time algorithms on planar graphs are significantly faster than the algorithms for general graphs: for example, 3-Coloring can be solved in time $2^{O(\sqrt{n})}$ in an n-vertex planar graph, whereas only $2^{O(n)}$-time algorithms are known for general graphs. For various planar problems, we often see a square root appearing in the running time of the best algorithms, e.g., the running time is often of the form $2^{O(\sqrt{n})}$, $n^{O(\sqrt{k})}$, or $2^{O(\sqrt{k})} \cdot n$. By now, we have a good understanding of why this square root appears. On the algorithmic side, most of these algorithms rely on the notion of treewidth and its relation to grid minors in planar graphs (but sometimes this connection is not obvious and takes some work to exploit). On the lower bound side, under a complexity assumption called Exponential Time Hypothesis (ETH), we can show that these algorithms are essentially best possible, and therefore the square root has to appear in the running time.

* Research supported by the European Research Council (ERC) grant 280152.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, p. 1, 2013.

An Algorithm for Determining Whether a Pair of Polygons Is Reversible

Jin Akiyama[1] and Hyunwoo Seong[2]

[1] Research Center for Math Education, Tokyo University of Science,
1-3 Kagurazaka, Shinjuku, Tokyo 162-8601, Japan
ja@jin-akiyama.com
[2] Department of Mathematics, The University of Tokyo,
3-8-1 Komaba, Meguro, Tokyo 153-8914, Japan
hwseong@hotmail.com

Background of our talk is tessellabilities, reversibilities, and decomposabilities of polygons, polyhedra, and polytopes, where by the word "tessellability", we mean the capability of the polytope to tessellate. Although these three concepts seem quite different, but there is a strong connection linking them. These connections will be shown when we consider the lattices of tilings in $\mathbb{R}^2$ and tessellations in $\mathbb{R}^3$. In this talk, we mainly discuss reversibilties of polygons from the standpoint of algorithm. We give an algorithm to check whether a given pair of polygons α and β with the same area is reversible or not. Many old and new results together with various research problems relating this topic will be presented.

A given pair of polygons α and β is said to be **reversible** if α has a dissection along the edges of a dissection tree into a finite number of pieces which can be rearranged to form β under the following conditions:

(i) the entire perimeter of α gets into the interior of β,
(ii) the dissection tree of α does not include any vertex of α, and
(iii) the same dissection requirements are also imposed on β.

A polygon α is said to be **reversible** if there is a convex polygon β such that the pair α and β is reversible.

Let a pair A and B be reversible and $\boldsymbol{T(A,B)}$ be a **superimposition** of **tilings** $\boldsymbol{T(A)}$ and $\boldsymbol{T(B)}$ made by repeating reversions between A and B.

We define a **reversion trunk** $\boldsymbol{R(A)}$ as the convex hull of the terminal points of a dissection tree $T(B) \cap A$ of A. Note that a reversible polygon A can have various kinds of reversion trunks, depending on the superimposition $T(A, B)$. We denote by $\boldsymbol{\mathfrak{R}(A)}$ the set of all reversion trunks $R(A)$. Reversion trunk $R(B)$ and the set $\boldsymbol{\mathfrak{R}(B)}$ of all reversion trunks $R(B)$ are defined analogously.

Let $\{\hat{i}, \hat{j}\}$ be an orthonormal basis for the plane $\mathbb{R}^2$ and set a orthonormal coordinate system by $\{\hat{i}, \hat{j}\}$.

Let $f : \mathbb{R}^2 \to PU_2''$ (the set of all parallelograms with unit area) be the function which transforms $\boldsymbol{u} = u_x\hat{i} + u_y\hat{j} \in \mathbb{R}^2$ to the parallelogram with unit area which is similar to the parallelogram whose sides are determined by $\hat{i}$ and $\boldsymbol{u}$.

Let $\boldsymbol{P} \equiv \mathbb{R}^2_I \setminus (\{0\} \times [0, 1)) \subset \mathbb{R}^2$, where $\mathbb{R}^2_I$ means the first quadrant including axes.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 2–3, 2013.

Denote by $\mathfrak{D}(A)$ the inverse image $(f|_P)^{-1}(\mathfrak{R}(A))$ on the plane $\mathbb{R}^2$.

For polygons $A, B \in \mathrm{QP}$ (the set of all quasi-parallelogons), a relation $A \sim B$ means that the pair A and B is reversible.

Proposition 1. *For given reversible polygons A and B,*
$A \sim B \iff \mathfrak{R}(A) \cap \mathfrak{R}(B) \neq \emptyset \iff \mathfrak{D}(A) \cap \mathfrak{D}(B) \neq \emptyset$.

Theorem 1. *A decision algorithm to check whether a pair of polygons is reversible or not is as follows:*

1. For given two polygons A and B, first we check whether both polygons are reversible (i.e., quasi-parallelogon) and have the same area. Otherwise, the pair A and B can never be reversible.

2. Let the area of A and B be 1. Compute the sets $\mathfrak{D}^*(A)$ and $\mathfrak{D}^*(B)$ which are sets of points and horizontal open line segments on the plane $\mathbb{R}^2$.

3. Reflect the points of $\mathfrak{D}^*(A)$ and $\mathfrak{D}^*(B)$ on the second and third quadrants $\mathbb{R}^2_{II} \cup \mathbb{R}^2_{III}$ through the origin and then reflect the resultant points on the fourth quadrant $\mathbb{R}^2_{IV}$ or the interval $\{0\} \times [0, 1)$ across the y-axis to obtain $\mathfrak{D}(A) = (f|_P)^{-1}(R(A))$ and $\mathfrak{D}(B) = (f|_P)^{-1}(R(B))$, respectively.

4. If $\mathfrak{D}(A)$ and $\mathfrak{D}(B)$ has any intersection, the pair A and B is reversible. Otherwise, the pair can never be reversible.

References

1. Bolyai, F.: Tentamen juventutem. Marcos Vasarhelyini: Typis Collegii Refomatorum per Josephum et Simeonem Kali (1832) (in Latin)
2. Gerwien, P.: Zerschneidung jeder bliebigen Anzahl von gleichen geradlinigen Figuren in dieselben Stücke. Journal für die reine und angewandte Mathematik (Crelle's Journal) 10, 228–234 and Taf. III
3. Hilbert, D.: Mathematische Probleme. Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse. Subsequently in Bulletin of the American Mathematical Society 8, 437–479 (1901-1902)
4. Dehn, M.: Über den Rauminhalt, Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse, 345-54. Subsequently in Mathematische Annalen 55(3), 465–478 (1902)
5. Dudeney, H.: The Canterbury Puzzles. Thomas Nelson and Sons (1907)
6. Boltyanskii, V.G.: Equivalent and Equidecomposable Figures, D. C. Health and Co. (1963), Henn, A.K., Watts, C.E., Translated and adapted from the first Russian edition (1956)
7. Boltyanskii, V.G.: Hilbert's Third Problem, V. H. Winston and Sons. In: Silverman, A., Translated by from the first Russian edition (1978)
8. Akiyama, J., Nakamura, G.: Congruent dissections of triangles and quadrilaterals - All the hinge points are on the sides of the polygon. In: Aronov, B., Basu, S., Pach, J., Sharir, M. (eds.) Discrete and Computational Geometry, The Goodman-Pollak Festschrift. Algorithms and Combinatorics, pp. 43–63. Springer (2003)
9. Akiyama, J., Sato, I., Seong, H.: On reversibility among parallelohedra. In: Márquez, A., Ramos, P., Urrutia, J. (eds.) EGC 2011. LNCS, vol. 7579, pp. 14–28. Springer, Heidelberg (2012)

Disjoint Small Cycles in Graphs and Bipartite Graphs*

Ding Ma and Yunshu Gao

School of Mathematics and Computer Science, Ningxia University, Yinchuan,750021, China
gysh2004@gmail.com

Abstract. Let k be a positive integer and let G be a graph of order $n \geq 3k+1$, X be a set of any k distinct vertices of G. It is proved that if $d(x) + d(y) \geq n + 2k - 2$ for any pair of nonadjacent vertices $x, y \in V(G)$, then G contains k disjoint cycles $T_1, \cdots, T_k$ such that each cycle contains exactly one vertex in X, and $|T_i| = 3$ for each $1 \leq i \leq k$ or $|T_k| = 4$ and the rest are all triangles. We also obtained two results about disjoint 6-cycles in a bipartite graph.

Keywords: Degree condition, Vertex-disjoint, Triangles.

1 Introduction

In this paper, we only consider finite undirected graphs without loops or multiple edges and we use Bondy and Murty [1] for terminology and notation not defined here. Let $G = (V, E)$ be a graph, the order of G is $|G| = |V|$ and its size is $E(G) = |E|$. A set of subgraphs is said to be vertex-disjoint or independent if no two of them have any common vertex in G, and we use disjoint or independent to stand for vertex-disjoint throughout this paper. Let G_1 and G_2 be two subgraphs of G or a subsets of $V(G)$. If G_1 and G_2 have no any common vertex in G, we define $E(G_1, G_2)$ to be the set of edges of G between G_1 and G_2, and let $E(G_1, G_2) = |E(G_1, G_2)|$. Let H be a subgraph of G and $u \in V(G)$ a vertex of G, $N(u, H)$ is the set of neighbors of u contained in H. We let $d(u, H) = |N(u, H)|$. Clearly, $d(u, G)$ is the degree of u in G, we often write $d(x)$ to replace $d(x, G)$. The minimum degree of G will be denoted by $\delta(G)$. If there is no fear of confusion, we often identify a subgraph H of G with its vertex set $V(H)$, for a vertex $x \in V(G) - V(H)$, we also denote $N_H(x) = N_G(x) \cap V(H)$ and $d_H(x) = |N_H(x)|$. For a subset U of $V(G)$, $G[U]$ denotes the subgraph of G induced by U. If H is a subgraph in G, we define $d_H(U) = \sum_{x \in U} d_H(x)$. Let C and P be a cycle and a path, respectively, we use $l(C)$ and $l(P)$ to denote the length of C and P, respectively. That is, $l(C) = |C|$ and $l(P) = |P| - 1$. A Hamiltonian cycle of G is a cycle which contains all vertices of G, and a Hamiltonian path of G is a path of G which contains every vertex in G. Let $v_1, \ldots, v_k$ be k distinct vertices in G, and let $C_1, \ldots, C_k$ be k disjoint cycles passing through $v_1, \ldots v_k$, respectively, in G. Then we say that G has k disjoint cycles $C_1, \ldots, C_k$ with respect to $\{v_1, \ldots v_k\}$. A cycle of

* Supported by the National Natural Science Foundation of China (Grant No. 11161035), Ningxia Ziran (Grant No. NZ1153) and research grant from Ningxia University (Grant No. ndzr10-19).

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 4–11, 2013.

length 3 is called a triangle and a cycle of length 4 is called a quadrilateral. For a cycle C with $l(C) = k$, we call that C be a k-cycle. Let v be a vertex and H is a subgraph of G, we say H is a v-subgraph if $v \in V(H)$. In particular, a v-cycle or a v-path is a cycle or path passing through v, respectively. For a graph G, we define

$$\sigma_2(G) = \min\{d(x) + d(y) \,|xy \notin E(G)\}$$

When G is a complete graph, we define $\sigma_2(G) = \infty$.In 1963, Corrádi and Hajnal [2] proved Erdös's conjecture in the early 1960s which concerns independent cycles in a graph. They proved that if G is a graph of order $n \geq 3k$ with $\delta(G) \geq 2k$, then G contains k disjoint cycles. In particular, when the order of G is exactly $3k$, then G contains k disjoint triangles. In the same year, Dirac [3] obtained the following result.

Theorem 1. *(Dirac [3]) Let k, n be two positive integers and let G be a graph of order $n \geq 3k$. If $\delta(G) \geq (n+k)/2$, then G contains k independent triangles.*

Many people have studied the problems of cyclability, which concerns that for a given subset S of vertices, whether there exists a cycle or several independent cycles that covering S.Which motivated us to be interested in the following problem, for any k independent vertices $v_1, \ldots, v_k$, what ensures that there exist k disjoint triangles $C_1, \ldots, C_k$ with respect to $\{v_1, \ldots, v_k\}$, such that each C_i ($i \in \{1, 2, \ldots, k\}$) contains exactly one vertex of v_i ($i \in \{1, 2, \ldots, k\}$). For the disjoint triangles covering, Li et al. [6] have obtained the following result.

Theorem 2. *(H.Li [6]) Let k, n be two positive integers and let G be a graph of order $n \geq 3k$, X a set of any k distinct vertices of G. If the minimum degree $\delta(G) \geq (n+2k)/2$, then G contains k disjoint triangles such that each triangle contains exactly one vertex of X.*

In this paper, we obtain the following result.

Theorem 3. *Let k be a positive integer and let G be a graph of order $n \geq 3k+1$, X a set of any k distinct vertices of G. If $\sigma_2(G) \geq n+2k-2$, then G contains k disjoint cycles $T_1, \ldots T_k$ such that each cycle contains exactly one vertex in X, and $|T_i| = 3$ for each $1 \leq i \leq k$ or $|T_k| = 4$ and the rest are all triangles.*

Remark 1. The condition $n \geq 3k+1$ in Theorem 1.3 is necessary since there exists a graph G with $|V(G)| \geq 4k$ and $\sigma_2(G) = |V(G)| + k - 1$ such that G does not even contain k vertex disjoint triangles (see [5]).

The remainder of this paper is organized as follows. In Section 2, we list several lemmas which will be used to prove Theorem 1.3 in Section 3. In Section 4, we obtain two results about disjoint 6-cycles in a bipartite graph and conclude this paper in Section 5 by proposing two related problems.

2 Lemmas

Lemma 1. *[4] Let $P = u_1u_2 \dots u_s\,(s \geq 2)$ be a path in G, $u \in V(G) - V(P)$, when $uu_1 \notin E(G)$, if $d(u,P) + d(u_s,P) \geq s$, then G has a path P' with vertex set $V(P') = V(P) \cup \{u\}$ whose end vertices are u and u_1. When $uu_1 \in E(G)$, if $d(u,P)+d(u_s,P) \geq s+1$, then G has a path P' with vertex set $V(P') = V(P)\cup\{u\}$ whose end vertices are u and u_1.*

Lemma 2. *[4] Let $P = u_1u_2 \dots u_s$ be a path with $s \geq 3$ in G. If $d(u_s,P) + d(u_1,P) \geq s$, then G has a cycle C with $V(C) = V(P)$.*

3 Proof of Theorem 1.3

Proof. Suppose that G does not contain k disjoint cycles $T_1, \dots, T_k$ such that each cycle contains exactly one vertex in X and $|T_k| = 4$ and the rest are triangles. We prove that G contains k disjoint triangles $T_1, \dots T_k$ such that each cycle contains exactly one vertex in X. Suppose this is false, let G be an edge-maximal counterexample. Since a complete graph of order $n \geq 3k+1$ contains k disjoint triangles such that each triangle contains exactly one vertex of X, thus, G is not a complete graph. Let u and v be nonadjacent vertices of G and define $G' = G + uv$, the graph obtained from G by adding the edge uv. Then G' is not a counterexample by the maximality of G, that is, for any $X = \{v_1, \dots, v_k\} \subseteq V(G)$, G' contains k disjoint triangles $T_1, \dots T_k$ with respect to $\{v_1, \dots, v_k\}$.

Claim. $k \geq 2$

Proof. Otherwise, suppose $k = 1$. By the classical result of Ore, G contains a Hamiltonian cycle $C = y_1y_2 \cdots y_ny_1$. We may assume that $v_1 = y_1$, otherwise, we can relabel the index of C.

We consider the path $P = y_1y_2y_3y_4$. Then $y_1y_3 \notin E(G)$, $y_2y_4 \notin E(G)$, $N(y_1, C - V(P)) \cap N(y_3, C - V(P)) = \phi$ and $N(y_2, C - V(P)) \cap N(y_4, C - V(P)) = \phi$. Then it follows that $2n \leq \Sigma_{x \in V(P)} d(x,G) \leq 6 + 2(n-4) = 2n-2$, a contradiction.

By the choice of G, there exists $v \in \{v_1, v_2 \dots, v_k\}$ such that G contains $k-1$ triangles $T_1, \dots T_{k-1}$ with respect to $\{v_1, v_2 \dots, v_k\} - \{v\}$, $v \notin V\left(\bigcup_{i=1}^{k-1} T_i\right)$. Subject to this, we choose $v \in \{v_1, v_2 \dots, v_k\}$ and $k-1$ triangles $T_1, \dots, T_{k-1}$ with respect to $\{v_1, v_2 \dots, v_k\} - \{v\}$ such that

$$\textit{The length of the longest v-path in } G - V\left(\bigcup_{i=1}^{k-1} T_i\right). \tag{1}$$

Let $P = u_1 \dots u_s$ be a longest v-path in $G - V\left(\bigcup_{i=1}^{k-1} T_i\right)$. Subject to (1), we choose $v \in \{v_1, v_2 \dots, v_k\}$, $k-1$ vertex disjoint triangles $T_1, \dots, T_{k-1}$ with respect to $\{v_1, v_2 \dots, v_k\} - \{v\}$ and P such that

$$\lambda(v, P). \tag{2}$$

Without loss of generality, suppose that $v = v_k$ and $v_i \in V(T_i)$ for each $i \in \{1, 2, \ldots, k-1\}$. Let $H = \bigcup_{i=1}^{k-1} T_i$, $D = G - H$ and $|D| = d$. Clearly, $d \geq 4$ as $n \geq 3k + 1$. Furthermore, by the choice of G, $s \geq 3$.

Claim. P is a Hamiltonian path of D.

Proof. Suppose $s \leq d$. We choose an arbitrary vertex $x_0 \in D - V(P)$. Clearly, $x_0u_1 \notin E(G)$ and $x_0u_s \notin E(G)$. Note $s \geq 3$, by (1) and Lemma 2.1, $d(x_0, P) + d(u_1, P) \leq s - 1$. Since $d(x_0, D - V(P)) \leq d - s - 1$ and $d(u_1, D - V(P)) = 0$, it follows that $d(x_0, D) + d(u_1, D) \leq d - 2$. By the assumption on the degree condition of G, we have

$$d(x_0, H) + d(u_1, H) \geq (n + 2k - 2) - (d - 2) = 5(k - 1) + 2$$

This implies that there exists $T_i \in H$, say T_1, such that $d(x_0, T_1) + d(u_1, T_1) = 6$. Let $T_1 = v_1w_1w_2v_1$. If we replace T_1 with $x_0w_2v_1x_0$, we obtain a path $P' = w_1u_1 \ldots u_s$ with $|P'| = |P| + 1$, contradicting (1).

Claim. If $\lambda(v_k, P) = 0$ or 1, then D is Hamiltonian.

Proof. By Claim 3.2, D contains a Hamiltonian path $P = u_1 \ldots u_d$ passing through v_k. If $u_1u_d \in E(G)$, then we have nothing to prove. So, $u_1u_d \notin E(G)$. By symmetry, if $\lambda(v_k, P) = 0$, we may assume that $v_k = u_1$. If $\lambda(v_k, P) = 1$, we assume that $v_k = u_2$.

If there exists $T_i \in H_T$ such that $d(u_1, T_i) + d(u_d, T_i) = 6$, then there exists $w \in V(T_i)$ with $u_1w \in E(G)$ such that $T_i - w + u_d$ contains a triangle T_i' passing through v_i. If we replace T_i with T_i', we see that D contains a v_k-path $P' = P - u_d + w$. However, $\lambda(v_k, P') = \lambda(v_k, P) + 1$, contradicting (2) while (1) still maintains. Hence,$d(u_1, T_i) + d(u_d, T_i) \leq 5$ for each $T_i \in H$ and so $d(u_1, H) + d(u_d, H) \leq 5(k - 1)$. It follows that

$$d(u_1, D) + d(u_d, D) \geq n + 2k - 2 - 5(k - 1) = d.$$

By Lemma 2.2, D contains a Hamiltonian cycle. This proves the claim.

Case 1. $d = 4$.

By Claim 3.3, D contains a Hamiltonian cycle C. Then G contains $k - 1$ disjoint triangles $T_1, T_2, \ldots, T_{k-1}$ and a quadrilateral C with respect to $\{v_1, v_2, \ldots, v_k\}$, a contradiction.

Case 2. $d \geq 5$.

By Claims 3.2 and 3.3, for each v_k-path P' of length 4 in D, we may assume that $\lambda(v_k, P') = 2$. Let $P' = y_1y_2y_3y_4y_5$ be an arbitrary v_k-path of length 4, then $v_k = y_3$. Since D does not contain a triangle passing through $y_3 = v_k$, hence, $y_1y_3 \notin E(G)$ and $y_2y_4 \notin E(G)$. Let $P'' = P' - y_5$. Since D contains no quadrilateral passing through $v_k = y_3$, then $N(y_1, D - V(P'')) \cap N(y_3, D - V(P'')) = \phi$ and

$N(y_2, D - V(P'')) \cap N(y_4, D - V(P'')) = \phi$. So, $\sum_{x \in V(P'')} d(x, D) \leq 2(d-4) + 6 = 2d - 2$. This gives that

$$\sum_{x \in V(P'')} d(x, H) \geq 2(n + 2k - 2) - (2d - 2) = 10(k-1) + 2$$

This implies that there exists $T_i \in H$, say T_1, such that $E(P'', T_1) \geq 11$. That is, there is at most one edge absent between P'' and T_1. Let $T_1 = v_1 w_1 w_2 v_1$.

We claim that $d(y_3, T_1) = 2$. Otherwise, say $d(y_3, T_1) = 3$. If $G[\{y_1, y_2, v_1\}]$ contains a triangle, denoted by $T_1{}'$, then G contains k disjoint triangles $T_1{}', T_2, \ldots T_{k-1}$, $y_3 w_1 w_2 y_3$ with respect to $\{v_1, v_2, \ldots, v_k\}$, a contradiction. Hence, $E(v_1, y_1 y_2) \leq 1$ and $E(P'', T_1) \leq 11$, which yields to $d(y_4, T_1) = 3$ and $y_2 w_2 \in E(G)$. Consequently, G contains k disjoint triangles $y_4 w_1 v_1 y_4, T_2, \ldots, T_{k-1}, y_2 y_3 w_2 y_2$ with respect to $\{v_1, v_2, \ldots, v_k\}$, a contradiction.

Since $d(y_3, T_1) = 2$, without loss of generality, say $y_3 w_2 \in E(G)$. Furthermore, we have $d(y_i, T_1) = 3$ for each $i \in \{1, 2, 4\}$. Then G contains k disjoint triangles $y_1 y_2 v_1 y_1, T_2, \ldots T_{k-1}$ and $y_3 y_4 w_2 y_3$ with respect to $\{v_1, v_2, \ldots, v_k\}$, a contradiction. This completes the proof of Case 2 and the proof of Theorem 3.

4 Bipartite Graph

In this section, we consider the disjoint 6-cycles in a bipartite graph. We list several useful lemmas.

Lemma 3. *Let C be a 6-cycle of G. Let $x \in V_1$ and $y \in V_2$ be two distinct vertices not on C. If $d(x, C) + d(y, C) \geq 5$, then there exists $z \in V(C)$ such that $C - z + x$ is a 6-cycle and $yz \in E(G)$.*

Proof. Without loss of generality, let $C = x_1 x_2 \ldots x_6 x_1$ with $x_1 \in V_1$. If $d(x, C) = 3$, since $d(y, C) \geq 2$, take any neighbor of $N(y, C)$ as z, the lemma is obvious. Hence, we may assume that $d(x, C) = 2$ and $d(y, C) = 3$. If $N(x, C) = \{x_2, x_4\}$, then $C - x_3 + x$ is a 6-cycle with $y x_3 \in E(G)$, we are done. By symmetry, we have $N(x, C) = \{x_2, x_6\}$. Then $C - x_1 + x$ is a 6-cycle with $y x_1 \in E(G)$. This proves the lemma.

Lemma 4. *[7] Let C be a 6-cycle, P_1, P_2 and P_3 be three paths in G with $l(P_1) = l(P_2) = l(P_3) = 1$. Suppose that C, P_1, P_2 and P_3 are disjoint and $E(C, P_1 \cup P_2 \cup P_3) \geq 13$, then $G[V(C \cup P_1 \cup P_2 \cup P_3)]$ contains a 6-cycle C' and a path P of order 6 such that C' and P are disjoint.*

Lemma 5. *[7] Let P_1 and P_2 be two disjoint paths in G with $l(P_1) = l(P_2) = 5$. If $E(P_1, P_2) \geq 7$, then $G[V(P_1 \cup P_2)]$ contains a 6-cycle.*

Lemma 6. *[7] Let C be a 6-cycle, P_1 and P_2 be two paths in G with $l(P_1) = l(P_2) = 5$. Suppose that C, P_1 and P_2 are disjoint and $E(C, P_1 \cup P_2) \geq 25$, then $G[V(C \cup P_1 \cup P_2)]$ contains two disjoint 6-cycles.*

Theorem 4. *Let k be a positive integer and $G = (V_1, V_2; E)$ a bipartite graph with $|V_1| = |V_2| = 3k$. Suppose $\delta(G) \geq 2k$, then G contains $k - 1$ disjoint 6-cycles and a path of order 6 such that all of them are disjoint.*

Proof. Suppose on the contrary, G does not contain k disjoint 6-cycles and a path of order 6. Let G be an edge-maximal counterexample. Since a complete bipartite graph with $|V_1| = |V_2| = 3k$ contains k disjoint 6-cycles and a path of order 6 such that all of them are disjoint. Thus, G is not a complete bipartite graph. Take any nonadjacent pair $u \in V_1$ and $v \in V_2$, $G+xy$ contains k disjoint 6-cycles and a path of order 6 such that all of them are disjoint. This implies G contains $k-2$ disjoint 6-cycle $Q_1, Q_2, \ldots, Q_{k-1}$ and a subgraph D. We divide the proof into two cases:

Case 1. D contains a 6-cycle, denoted by Q_k.

Let $H = \cup_{i=1}^{k} Q_i$ and $D = G - V(H)$. We may assume that D contains at least one edge. Otherwise, take any pair of $u \in V_1 \cap D$ and $v \in V_2 \cap D$. Then $uv \notin E(G)$ and $d(u, D) + d(v, D) = 0$. Consequently, $d(u, H) + d(v, H) \geq 4(k-1) + 4$, which implies that there exists $Q_i \in H$ such that $d(u, Q_i) + d(v, Q_i) \geq 5$. By Lemma 3, $G[V(Q_i) \cup \{u, v\}]$ contains a 6-cycle $Q_i{}'$ and edge e such that $Q_i{}'$ and e are disjoint. Replace Q_i with $Q_i{}'$, we see D contains an edge.

Let uv be an edge in D with $u \in V_1 \cap D$. An argument analogous to the process of above can lead to D contains three disjoint edges. Denoted by uv, xy and ab with $\{u, x, a\} \subseteq V_1$. Now we will prove that D contains a path of order 6. Otherwise, $E(D) \leq 5$ and so we have

$$\sum_{x \in V(D)} d(x, H) \geq 12k - 10 = 12(k-1) + 2$$

This implies that there exists $Q_i \in H$ such that $\sum_{x \in V(D)} d(x, Q_i) \geq 13$. By Lemma 4, $G[V(Q_i \cup D)]$ contains a 6-cycle $Q_i{}'$ and a path P of order 6 such that C' and P are disjoint. Replace Q_i with $Q_i{}'$, we see that D contains a path of order 6, a contradiction.

Case 2. D contains two disjoint paths of order 6, denoted by P_1 and P_2.

In this case, G contains $k-2$ disjoint 6-cycles $Q_1, Q_{2,\ldots}Q_{k-2}$. Let $H' = \cup_{i=1}^{k-2} Q_i$. By Case 1, we may assume that $G[V(P_1 \cup P_2)]$ contains no 6-cycle. This leads to $E(P_1) \leq 7$ and $E(P_2) \leq 7$ and $E(P_1, P_2) \leq 6$ by Lemma 5. Consequently, we obtain

$$\sum_{x \in V(P_1 \cup P_2)} d(x, H) \geq 24k - 40 = 24(k-2) + 8$$

This implies that there exists $Q_i \in H'$ such that $\sum_{x \in V(P_1 \cup P_2)} d(x, Q_i) \geq 25$. By Lemma 6, $G[V(Q_i \cup P_1 \cup P_2)]$ contains two disjoint 6-cycles. By Case 1, we obtain a contradiction. This completes the proof of Theorem 4.

Theorem 5. *Let k be a positive integer and $G = (V_1, V_2; E)$ a bipartite graph with $|V_1| = |V_2| = 3k$. Suppose $\delta(G) \geq 2k$, then G contains k disjoint 6-cycles or $k-1$ disjoint 6-cycles and a quadrilateral such that all of them are disjoint.*

Proof. Suppose that G does not contain $k-1$ disjoint 6-cycles and a quadrilateral such that all of them are disjoint, we will show that G contains k disjoint 6-cycles. Suppose that this is not true. Let G be a edge-maximal counterexample. That is, for any pair

of nonadjacent vertices $u \in V_1$ and $v \in V_2$, $G + uv$ contains k disjoint 6-cycles. This implies G contains $k-1$ disjoint 6-cycles $Q_1, \ldots Q_{k-1}$ and a path P of order 6. Denote $H = \cup_{i=1}^{k} Q_i$.

Since $G[V(P)]$ does not contain a quadrilateral or a 6-cycle, we see that $E(P) = 5$. Therefore, $\sum_{x \in V(P)} d(x, H) \geq 12k - 10 = 12(k-1) + 2$. This implies that there exists $Q_i \in H$ such that $\sum_{x \in V(P)} d(x, Q_i) \geq 13$. By our assumption, $G[V(Q_i \cup P)]$ contains two disjoint 6-cycles or two disjoint cycles with one quadrilateral and the other 6-cycle.

Without loss of generality, say $Q_i = a_1 a_2 \ldots a_6 a_1$ and $P = x_1 x_2 \ldots x_6 x_1$ with $\{a_1, x_1\} \subseteq V_1$. Set $d(a_1, P) = \max\{d(a_i, P) \,|\, i \in \{1, 2, 3, 4, 5, 6\}\}$. Clearly, $d(a_1, P) \geq \left\lceil \frac{13}{6} \right\rceil = 3$, which means $N(x_1, P) = \{x_2, x_4, x_6\}$. Furthermore, we observe that $d(x_j, Q_i) \leq 1$ for each $j \in \{1, 3, 5\}$. Otherwise, without loss of generality, say $d(x_1, Q_i) \geq 2$. If $\{a_4, a_6\} \subseteq N(x_1, Q_i)$, then $G[V(Q_i \cup P)]$ contains a 6-cycle $P - x_1 + a_1$, which disjoint from a quadrilateral $x_1 a_4 a_5 a_6 x_1$, a contradiction. Hence, by symmetry, we may assume that $\{a_2, a_6\} \subseteq N(x_1, Q_i)$. Then $G[V(Q_i \cup P)]$ contains two disjoint 6-cycles $P - x_1 + a_1$ and $C - a_1 + x_1$, a contradiction. Consequently, it follows that $E(Q_i, P) \leq 3 + 3 + 6 = 12$, which contradicts the fact that $E(Q_i, P) \geq 13$, a final contradiction.

Remark 2. The degree condition in Theorem 4 is sharp in general. To see this, we construct bipartite graph G for positive integer as follows. Let $G_1 = (A, B; E_1)$ and $G_2 = (X, Y; E_2)$ be two independent complete bipartite graph with $|A| = |Y| = 2(k-1)$ and $|B| = |X| = k + 2$. Then G consists of G_1, G_1 and a set of $k + 2$ independent edges between B and X, and finally, join every vertex in A to every vertex in Y. Clearly, $(A \cup X, B \cup Y)$ is a bipartition of G. It is easy to see that G does not contain $k - 1$ disjoint 6-cycles and a path of order 6 such that all of them are disjoint. We see that the minimum degree of G is $2k - 1$.

5 Conjectures

To conclude this paper, we propose the following two conjectures for readers to discuss.

Conjecture 1. Let $k \geq 2$, n be two positive integers and let G be a graph of order $n \geq 3k + 1$, X a set of any k distinct vertices of G. If $\sigma_2(G) \geq n + 2k - 1$, G contains k disjoint triangles $T_1, \ldots T_k$ such that each triangle contains exactly one vertex in X.

Conjecture 2. Let k be a positive integer and $G = (V_1, V_2; E)$ a bipartite graph with $|V_1| = |V_2| = 3k$. Suppose $\delta(G) \geq 2k$, G contains k disjoint 6-cycles.

If Conjecture 2 is true, the degree condition is also sharp by Remark 2.

References

1. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. North-Holland, Amsterdam (1976)
2. Corrádi, K., Hajnal, A.: On the maximal number of independent circuits in a graph. Acta Math. Acad. Sci. Hungar 14, 423–439 (1963)

3. Dirac, G.A.: On the maximal number of independent triangles. Abh. Math. Debrecen 9, 78–82 (1963)
4. Dong, J.: k disjoint cycles containing specified independent vertices. Discrete Math. 308, 5269–5273 (2008)
5. Justesen, P.: On Independent Circuits in Finite Graphs and a Conjecture of Erdös and Pósa. Annals of Discrete Mathematics 41, 299–306 (1989)
6. Li, H., Li, J.: Independent triangle covering given vertices of a graph. Theoretical Computer Science 263, 333–344 (2001)
7. Zhu, S., Hao, R.: Disjoint 6-cycles in Bipartite Graphs. Advances in Mathematics 36, 617–626 (2007)
8. Psa, L.: On The Circuits Of Finite Graphs. Magyar Tud. Akad. Mat. Kutató Int. Kőzl 8, 355–361 (1964)
9. Wang, H.: On The Maximum Number Of Independent Cycles In A Graph. Discrete Math. 205, 183–190 (1999)
10. Lesniak, L.: Independent Cycles In Graphs. J. Combin. Math. Combin. Comput. 17, 55–63 (1995)

An Algorithm for Listing All Minimal 2-Dominating Sets of a Tree

Marcin Krzywkowski*

Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology
Poland
marcin.krzywkowski@gmail.com

Abstract. We provide an algorithm for listing all minimal 2-dominating sets of a tree of order n in time $\mathcal{O}(1.3248^n)$. This implies that every tree has at most 1.3248^n minimal 2-dominating sets. We also show that this bound is tight.

Keywords: domination, 2-domination, minimal 2-dominating set, tree, combinatorial bound, exponential algorithm, listing algorithm.

1 Introduction

Let $G = (V, E)$ be a graph. The order of a graph is the number of its vertices. By the neighborhood of a vertex v of G we mean the set $N_G(v) = \{u \in V(G): uv \in E(G)\}$. The degree of a vertex v, denoted by $d_G(v)$, is the cardinality of its neighborhood. By a leaf we mean a vertex of degree one, while a support vertex is a vertex adjacent to a leaf. The distance between two vertices of a graph is the number of edges in a shortest path connecting them. The eccentricity of a vertex is the greatest distance between it and any other vertex. The diameter of a graph G, denoted by $\mathrm{diam}(G)$, is the maximum eccentricity among all vertices of G. Denote by P_n a path on n vertices. By a star we mean a connected graph in which exactly one vertex has degree greater than one.

A subset $D \subseteq V(G)$ is a dominating set of G if every vertex of $V(G) \setminus D$ has a neighbor in D, while it is a 2-dominating set of G if every vertex of $V(G) \setminus D$ has at least two neighbors in D. A dominating (2-dominating, respectively) set D is minimal if no proper subset of D is a dominating (2-dominating, respectively) set of G. A minimal 2-dominating set is abbreviated as m2ds. Note that 2-domination is a type of multiple domination in which each vertex, which is not in the dominating set, is dominated at least k times for a fixed positive integer k. Multiple domination was introduced by Fink and Jacobson [7], and further studied for example in [2,10,18]. For a comprehensive survey of domination in graphs, see [11,12].

* The research was supported by the Polish National Science Centre grant 2011/03/N/ST6/04404.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 12–16, 2013.

Observation 1. *Every leaf of a graph G is in every 2-dominating set of G.*

One of the typical questions in graph theory is how many subgraphs of a given property can a graph on n vertices have. For example, the famous Moon and Moser theorem [17] says that every graph on n vertices has at most $3^{n/3}$ maximal independent sets.

Combinatorial bounds are of interest not only on their own, but also because they are used for algorithm design as well. Lawler [16] used the Moon-Moser bound on the number of maximal independent sets to construct an $(1 + \sqrt[3]{3})^n \cdot n^{\mathcal{O}(1)}$ time graph coloring algorithm, which was the fastest one known for twenty-five years. In 2003 Eppstein [6] reduced the running time of a graph coloring to $\mathcal{O}(2.4151^n)$. In 2006 the running time was reduced [1,14] to $\mathcal{O}(2^n)$. For an overview of the field, see [9].

Fomin et al. [8] constructed an algorithm for listing all minimal dominating sets of a graph on n vertices in time $\mathcal{O}(1.7159^n)$. There were also given graphs ($n/6$ disjoint copies of the octahedron) having $15^{n/6} \approx 1.5704^n$ minimal dominating sets. This establishes a lower bound on the running time of an algorithm for listing all minimal dominating sets of a given graph.

The number of maximal independent sets in trees was investigated in [19]. Couturier et al. [5] considered minimal dominating sets in various classes of graphs. The authors of [13] investigated the enumeration of minimal dominating sets in graphs.

Bród and Skupień [3] gave bounds on the number of dominating sets of a tree. They also characterized the extremal trees. The authors of [4] investigated the number of minimal dominating sets in trees containing all leaves.

In [15] an algorithm was given for listing all minimal dominating sets of a tree of order n in time $\mathcal{O}(1.4656^n)$, implying that every tree has at most 1.4656^n minimal dominating sets. An infinite family of trees for which the number of minimal dominating sets exceeds 1.4167^n was also given. This established a lower bound on the running time of an algorithm for listing all minimal dominating sets of a given tree.

We provide an algorithm for listing all minimal 2-dominating sets of a tree of order n in time $\mathcal{O}(1.3248^n)$. This implies that every tree has at most 1.3248^n minimal 2-dominating sets. We also show that this bound is tight.

2 Results

We describe an algorithm for listing all minimal 2-dominating sets of a given input tree. We prove that the running time of the algorithm is $\mathcal{O}(1.3248^n)$, implying that every tree has at most 1.3248^n minimal 2-dominating sets.

Theorem 2. *Every tree T of order n has at most α^n minimal 2-dominating sets, where $\alpha \approx 1.32472$ is the positive solution of the equation $x^3 - x - 1 = 0$.*

Proof. In our algorithm, the iterator of the solutions for a tree T is denoted by $\mathcal{F}(T)$. To obtain the upper bound on the number of minimal 2-dominating

sets of a tree, we prove that the algorithm lists these sets in time $\mathcal{O}(1.3248^n)$. Notice that the diameter of a tree can easily be determined in polynomial time. If $\operatorname{diam}(T) = 0$, then $T = P_1 = v_1$. Let $\mathcal{F}(T) = \{\{v_1\}\}$. Obviously, $\{v_1\}$ is the only m2ds of the path P_1. We have $n = 1$ and $|\mathcal{F}(T)| = 1$. We also have $1 < \alpha$. If $\operatorname{diam}(T) = 1$, then $T = P_2 = v_1v_2$. Let $\mathcal{F}(T) = \{\{v_1, v_2\}\}$. It is easy to observe that $\{v_1, v_2\}$ is the only m2ds of the path P_2. We have $n = 2$ and $|\mathcal{F}(T)| = 1$. Obviously, $1 < \alpha^2$. If $\operatorname{diam}(T) = 2$, then T is a star. Denote by x the support vertex of T. Let $\mathcal{F}(T) = \{V(T) \setminus \{x\}\}$. It is easy to observe that $V(T) \setminus \{x\}$ is the only m2ds of the tree T. We have $n \geq 3$ and $|\mathcal{F}(T)| = 1$. Obviously, $1 < \alpha^n$.

Now consider trees T with $\operatorname{diam}(T) \geq 3$. The results we obtain by the induction on the number n. Assume that they are true for every tree T' of order $n' < n$. The tree T can easily be rooted at a vertex r of maximum eccentricity $\operatorname{diam}(T)$ in polynomial time. A leaf, say t, at maximum distance from r, can also be easily computed in polynomial time. Let v denote the parent of t and let u denote the parent of v in the rooted tree. If $\operatorname{diam}(T) \geq 4$, then let w denote the parent of u. By T_x we denote the subtree induced by a vertex x and its descendants in the rooted tree T.

If $d_T(v) \geq 3$, then let $T' = T - T_v$ and let T'' differ from T' only in that it has the vertex v. Let $\mathcal{F}(T)$ be as follows,

$$\begin{aligned}&\{D' \cup V(T_v) \setminus \{v\} \colon D' \in \mathcal{F}(T')\}\\ \cup\, &\{D'' \cup V(T_v) \setminus \{v\} \colon D'' \in \mathcal{F}(T'') \text{ and } D'' \setminus \{v\} \notin \mathcal{F}(T')\}.\end{aligned}$$

Let us observe that all elements of $\mathcal{F}(T)$ are minimal 2-dominating sets of the tree T. Now let D be any m2ds of T. Observation 1 implies that $V(T_v)\setminus\{v\} \subseteq D$. If $v \notin D$, then observe that $D \cap V(T')$ is an m2ds of the tree T'. By the inductive hypothesis we have $D \cap V(T') \in \mathcal{F}(T')$. Now assume that $v \in D$. It is easy to observe that $D \cap V(T'')$ is an m2ds of the tree T''. By the inductive hypothesis we have $D \cap V(T'') \in \mathcal{F}(T'')$. The set $D \cap V(T')$ is not an m2ds of the tree T', otherwise $D \setminus \{v\}$ is a 2-dominating set of the tree T, a contradiction to the minimality of D. By the inductive hypothesis we have $D \cap V(T') \notin \mathcal{F}(T')$. Therefore $\mathcal{F}(T)$ contains all minimal 2-dominating sets of the tree T. Now we get $|\mathcal{F}(T)| = |\mathcal{F}(T')| + |\{D'' \in \mathcal{F}(T'') \colon D'' \setminus \{v\} \notin \mathcal{F}(T')\}| \leq |\mathcal{F}(T')| + |\mathcal{F}(T'')|$ $\leq \alpha^{n-3} + \alpha^{n-2} = \alpha^{n-3}(\alpha + 1) = \alpha^{n-3} \cdot \alpha^3 = \alpha^n$.

If $d_T(v) = 2$ and $d_T(u) \geq 3$, then let $T' = T - T_v$, $T'' = T - T_u$, and

$$\mathcal{F}(T) = \{D' \cup \{t\} \colon u \in D' \in \mathcal{F}(T')\} \cup \{D'' \cup V(T_u) \setminus \{u\} \colon D'' \in \mathcal{F}(T'')\}.$$

Let us observe that all elements of $\mathcal{F}(T)$ are minimal 2-dominating sets of the tree T. Now let D be any m2ds of T. By Observation 1 we have $t \in D$. If $v \notin D$, then $u \in D$ as the vertex v has to be dominated twice. Observe that $D \setminus \{t\}$ is an m2ds of the tree T'. By the inductive hypothesis we have $D \setminus \{t\} \in \mathcal{F}(T')$. Now assume that $v \in D$. We have $u \notin D$, otherwise $D\setminus\{v\}$ is a 2-dominating set of the tree T, a contradiction to the minimality of D. Observe that $D \cap V(T'')$ is an m2ds of the tree T''. By the inductive hypothesis we have $D \cap V(T'') \in \mathcal{F}(T'')$. Therefore $\mathcal{F}(T)$ contains all minimal 2-dominating sets of the tree T. Now we get $|\mathcal{F}(T)| = |\{D' \in \mathcal{F}(T') \colon u \in D'\}| + |\mathcal{F}(T'')| \leq |\mathcal{F}(T')| + |\mathcal{F}(T'')| \leq \alpha^{n-2} + \alpha^{n-3}$ $= \alpha^{n-3}(\alpha + 1) = \alpha^{n-3} \cdot \alpha^3 = \alpha^n$.

If $d_T(v) = d_T(u) = 2$, then let $T' = T - T_v$, $T'' = T - T_u$, and

$$\mathcal{F}(T) = \{D' \cup \{t\} \colon D' \in \mathcal{F}(T')\} \cup \{D'' \cup \{v,t\} \colon w \in D'' \in \mathcal{F}(T'')\}.$$

Let us observe that all elements of $\mathcal{F}(T)$ are minimal 2-dominating sets of the tree T. Now let D be any m2ds of T. By Observation 1 we have $t \in D$. If $v \notin D$, then observe that $D \setminus \{t\}$ is an m2ds of the tree T'. By the inductive hypothesis we have $D \setminus \{t\} \in \mathcal{F}(T')$. Now assume that $v \in D$. We have $u \notin D$, otherwise $D \setminus \{v\}$ is a 2-dominating set of the tree T, a contradiction to the minimality of D. Moreover, we have $w \in D$ as the vertex u has to be dominated twice. Observe that $D \setminus \{v,t\}$ is an m2ds of the tree T''. By the inductive hypothesis we have $D \setminus \{v,t\} \in \mathcal{F}(T'')$. Therefore $\mathcal{F}(T)$ contains all minimal 2-dominating sets of the tree T. Now we get $|\mathcal{F}(T)| = |\mathcal{F}(T')| + |\{D'' \in \mathcal{F}(T'') \colon w \in D''\}|$ $\leq |\mathcal{F}(T')| + |\mathcal{F}(T'')| \leq \alpha^{n-2} + \alpha^{n-3} = \alpha^{n-3}(\alpha + 1) = \alpha^{n-3} \cdot \alpha^3 = \alpha^n$.

We now show that paths attain the bound from the previous theorem.

Proposition 3. *For positive integers n, let a_n denote the number of minimal 2-dominating sets of the path P_n. We have*

$$a_n = \begin{cases} 1 & \text{if } n \leq 3; \\ a_{n-3} + a_{n-2} & \text{if } n \geq 4. \end{cases}$$

Proof. It is easy to see that a path on at most three vertices has exactly one minimal 2-dominating set. Now assume that $n \geq 4$. Let $T' = T - v_n - v_{n-1}$ and $T'' = T' - v_{n-2}$. It follows from the last paragraph of the proof of Theorem 2 that $a_n = a_{n-3} + a_{n-2}$.

Solving the recurrence $a_n = a_{n-3} + a_{n-2}$, we get $\lim_{n\to\infty} \sqrt[n]{a_n} = \alpha$, where $\alpha \approx 1.3247$ is the positive solution of the equation $x^3 - x - 1 = 0$. This implies that the bound from Theorem 2 is tight.

It is an open problem to prove the tightness of an upper bound on the number of minimal dominating sets of a tree. In [15] it has been proved that any tree of order n has less than 1.4656^n minimal dominating sets. A family of trees having more than 1.4167^n minimal dominating sets has also been given.

References

1. Björklund, A., Husfeldt, T.: Inclusion-exclusion algorithms for counting set partitions. In: Proceedings of FOCS, pp. 575–582 (2006)
2. Blidia, M., Favaron, O., Lounes, R.: Locating-domination, 2-domination and independence in trees. Australasian Journal of Combinatorics 42, 309–316 (2008)
3. Bród, D., Skupień, Z.: Trees with extremal numbers of dominating sets. Australasian Journal of Combinatorics 35, 273–290 (2006)
4. Bród, D., Włoch, A., Włoch, I.: On the number of minimal dominating sets including the set of leaves in trees. International Journal of Contemporary Mathematical Sciences 4, 1739–1748 (2009)

5. Couturier, J.-F., Heggernes, P., van't Hof, P., Kratsch, D.: Minimal dominating sets in graph classes: combinatorial bounds and enumeration. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 202–213. Springer, Heidelberg (2012)
6. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. Journal of Graph Algorithms and Applications 7, 131–140 (2003)
7. Fink, J., Jacobson, M.: n-domination in graphs. In: Graph Theory with Applications to Algorithms and Computer Science, pp. 282–300. Wiley, New York (1985)
8. Fomin, F., Grandoni, F., Pyatkin, A., Stepanov, A.: Combinatorial bounds via measure and conquer: bounding minimal dominating sets and applications. ACM Transactions on Algorithms 5, article 9, 17 (2009)
9. Fomin, F., Kratsch, D.: Exact Exponential Algorithms. Springer, Berlin (2010)
10. Fujisawa, J., Hansberg, A., Kubo, T., Saito, A., Sugita, M., Volkmann, L.: Independence and 2-domination in bipartite graphs. Australasian Journal of Combinatorics 40, 265–268 (2008)
11. Haynes, T., Hedetniemi, S., Slater, P.: Fundamentals of Domination in Graphs. Marcel Dekker, New York (1998)
12. Haynes, T., Hedetniemi, S., Slater, P. (eds.): Domination in Graphs: Advanced Topics. Marcel Dekker, New York (1998)
13. Kanté, M., Limouzy, V., Mary, A., Nourine, L.: Enumeration of minimal dominating sets and variants. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 298–309. Springer, Heidelberg (2011)
14. Koivisto, M.: An $\mathcal{O}(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), pp. 583–590. IEEE (2006)
15. Krzywkowski, M.: Trees having many minimal dominating sets. Information Processing Letters 113, 276–279 (2013)
16. Lawler, E.: A note on the complexity of the chromatic number problem. Information Processing Letters 5, 66–67 (1976)
17. Moon, J., Moser, L.: On cliques in graphs. Israel Journal of Mathematics 3, 23–28 (1965)
18. Shaheen, R.: Bounds for the 2-domination number of toroidal grid graphs. International Journal of Computer Mathematics 86, 584–588 (2009)
19. Wilf, H.: The number of maximal independent sets in a tree. SIAM Journal on Algebraic and Discrete Methods 7, 125–130 (1986)

Algorithms for Testing Length Four Permutations

Yijie Han[1] and Sanjeev Saxena[2]

[1] School of Computing and Engineering, University of Missouri at Kansas City, Kansas City, MO 64110, USA
hanyij@umkc.edu
[2] Dept. of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India-208 016
ssax@iitk.ac.in

Abstract. We present new $\theta(n)$ time algorithms for testing pattern involvement for all length 4 permutations. For most of these permutations the previous best algorithms require $O(n \log n)$ time.

Keywords: Separable permutations, pattern matching, optimal algorithms.

1 Introduction

Pattern containment (also called pattern involvement) is a well studied problem in both Computer Science and Combinatorics [1] (see [7] for a survey of results).

Two permutations $P = p_1 p_2 ... p_k$ and $P' = p'_1 p'_2 ... p'_k$ are said to be order isomorphic if their letters are in the same relative order, i.e., $p_i < p_j$, if and only if, $p'_i < p'_j$. For example permutation 1, 3, 2, 4 is order isomorphic to 7, 19, 15, 23.

A permutation $P = p_1 p_2 ... p_k$ is said to be present in (or is involved in) another permutation $P' = p'_1 p'_2 ... p'_n$ if P' has a subsequence which is order isomorphic to P. The p_i's need not be consecutive in P'.

The general problem of testing presence of one permutation in another permutation is NP-complete [4]. However polynomial time algorithms are known when

1. $P = 1, 2, ..., k$. This becomes largest increasing subsequence problem [5].
2. P' is separable [4]. A permutation is separable if it contains pattern 2, 4, 1, 3 or its reverse 3, 1, 4, 2.
3. k is a constant. Brute force algorithm will take $O(n^k)$ time.

In the case when $k = 3$, linear time algorithms are possible [1]. And for the case $k = 4$, [1] have shown that $O(n \log n)$ time is possible. Further, a linear time algorithm exists to test whether a pattern 2, 4, 1, 3 (or its reverse 3, 1, 4, 2) is present; this is basically a test to check whether a pattern is separable [4].

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 17–23, 2013.

Linear time algorithms are also known for monotone patterns 1, 2, 3, 4 and (its reverse) 4, 3, 2, 1 [5].

Albert et.al. [1] studied the general problem of permutation involvement and proposed $O(n \log n)$ time algorithms for patterns of length 4. Their algorithms use orthogonal range queries of the kind:

Find the smallest number larger than a query item x between positions p and q.

As general orthogonal range queries take $O(\log n)$ query time after $O(n \log n)$ preprocessing time [8] (Theorem 2.12), $O(n \log n)$ time algorithms appear to be the best possible using this approach.

Our improvement comes firstly from use of the "usual" range maxima (minima) queries (instead of orthogonal range queries) of the kind:

Find the largest (smallest) number between positions p and q.

And for the case 1,3,2,4 we use a particular kind of analysis to achieve $\theta(n)$ time.

In this paper, we describe new linear time algorithms for all patterns of length 4.

Some useful "tools" are described in Section 2. Most cases of length 4 patterns are covered in Section 3. In Section 4 we describe the case 1,3,2,4.

2 Preliminaries

We give the definition for range minima and nearest largers problems. We will use routines for range minima and for nearest largers as black boxes.

For the *range minima* problem, we are given an array $A[1 : n]$, which we preprocess to answer queries of the form:

> Given two integers i, j with $1 \leq i \leq j \leq n$ the smallest item in sub-array $A[i : j]$.

Range minima queries take $O(1)$ time after $O(n)$ preprocessing cost [2,3]. Range maxima can also be solved in this way.

In the right *nearest largers problem*[3],

> for each item i of array $A[1 : n]$, we are to find $j > i$, closest to i, such that $A[j] > A[i]$ (thus items, $A[i + 1], A[i + 2], ..., A[j - 2], A[j - 1]$ are all smaller than $A[i]$). Or, $j = \min\{k | A[k] > A[i] \text{ and } k > i\}$.

The right nearest larger also take $O(1)$ time after $O(n)$ preprocessing cost [2,3]. All nearest largers can be found in $O(n)$ time [2,3]. The left nearest larger, the left nearest smaller, the right nearest smaller can also be solved in this way.

Let us assume that the permutation is given in array $P[1 : n]$. Thus, if the ith item of the pattern is k, then $P[i] = k$. As P is a permutation, all items of P are distinct. Hence, if $P[i] = k$ then we can define the *inverse mapping*

$$Position[k] = i$$

Thus, item $Y = P(y)$ will be to the right of item $X = P(x)$ in array P, if and only if, $y > x$ or equivalently $Position(Y) > Position(X)$. And item $Z = P(z)$ will be between X and Z if $Position(Z)$ is between (in value) $Position(X)$ and $Position(Y)$, i.e., $Position(X) < Position(Z) < Position(Y)$, or $Position(Y) < Position(Z) < Position(X)$

Let r be the nearest right larger of k in $Position$ array.

P-value	k	$k+1$	$k+2$	$\ldots$	r	$r+1$
$Position$					*	

Then, as items with P-value $k+1, ..., r-1$ are smaller, r is the first (smallest) P-value item larger than k and to its right. Moreover, the nearest smaller of k in the $Position$ array to the right is the first (smallest) P-value item larger than k and to its left.

Similarly, s the nearest larger of k in the $Position$ array on the left, is the first (largest) P-value item smaller than k and to its right. And nearest smaller of k in $Position$ array to its left, is the first (largest) P-value item smaller than k and to its left.

Thus,

Theorem 1. If we know $P[i] = k$, then we can find items closest in values (both larger and smaller than k) on either side of position i using nearest smallers or largers on the $Position$ array. □

We have:

Corollary 1. After preprocessing, if $P[i] = k$, then we can find items closest in values (both larger and smaller than k) on either side of position i in $O(1)$ time. The preprocessing time is $O(n)$ time. □

Let us assume that $P[i] = k$ and $P[j] = l$ with $k < l$.

P-value	k	$k+1$	$k+2$	$\ldots$	l	$l+1$
$Position$	i				j	

If x is the $RangeMinimaPosition(k, l)$ on $Position$-array, then x is the smallest (or the left most) $Position$-value between $Position(k)$ and $Position(l)$ of items with P-value between k and l.

Similarly, if y is the $RangeMaximaPosition(k, l)$ on $Position$-array, then y is the largest (or the right most) $Position$-value between $Position(k)$ and $Position(l)$ of items with P-value between k and l. Thus,

Theorem 2. Given any $P[i] = k$ and $P[j] = l$, we can find the left most and the right most items with P-values between $P[i]$ and $P[j]$ using range maxima or range minima queries. □

We have the following result:

Corollary 2. If $P[i] = k$, and $P[j] = l$, then we can find the left most and the right most items with P-values between k and l in $O(1)$ time after preprocessing. The preprocessing time is $O(n)$. □

3 Length Four Permutations Except 1324

For length 4 sequences, there will be 24 permutations, but 12 of these will be reverse (i.e., $P_0[i] = P[n-i+1]$) of some other. Further, 4 out of these 12 will be "complement" (i.e., $P[i] = n-P[i]+1$), thus in all 8 permutations will be left [1]:

1234, 2134, 2341, 2314, 3412, 2413, 1342, 1324

2413 (and its reverse 3142) is the case for separable permutation and linear time algorithm for only these are known [5]. In the next section we will give a linear time algorithm for the case 1324 (and its reverse 4231). Here we sketch how to deal with other permutations. As techniques for these permutations are similar, the description will be a bit brief.

Depending on the case, we try to see if i can be chosen as a "2" or a "3". We will abbreviate this to just "fix 2" (i.e, $i = i_2$) or "fix 3" (i.e., $i = i_3$). We finally get a witness for tuple (i_1, i_2, i_3, i_4), if $P[i_1] < P[i_2] < P[i_3] < P[i_4]$ and i_1, i_2, i_3, i_4 occur in the same order as $1, 2, 3, 4$. Again, a flag can be set if we have a witness and reset otherwise. Finally a logical "or" will give the answer.

In some of the cases, we have to search for an item (usually 3) which has a still larger item (which can be then chosen as 4). For this to be done efficiently, we define a new array $R[1:n]$. The element

$$R[i] = \begin{cases} P[i] \text{ if } i \text{ has a larger item to its right} \\ 0 \text{ otherwise} \end{cases}$$

By using right nearest largers, we can easily identify items which have a larger item to their right. Note that in R each nonzero element has a larger element to its right in P. Let us preprocess array R for range maxima queries.

The technique for various patterns is:

1234 Fix 2. 1 the smallest item to its left is $i_1 = RangeMinimaP(1, i)$. Item 3 can be obtained by range maxima on array R, $i_3 = RangeMaximaR(i, n)$. Finally, $i_4 = RangeMaximaP(i_3, n)$.

2134 Fix 2. Index i_1 of 1, the first item less than 2, can be found from right nearest smaller of i_2. And again 3 can be obtained by range maxima on array R, $i_3 = RangeMaximaR(i_1, n)$. Finally, $i_4 = RangeMaximaP(i_3, n)$.

2341 Fix 3. Index i_4 of 4, the first item more than 3, can be found from right nearest larger of i_3. $i_1 = RangeMinimaP(i_4, n)$ will choose 1 as the smallest item on the right of 4. And we use Corollary 1, to find i_2, the index of 2 as the item on left of i, just smaller than $P[i_3]$.

2314 Fix 3. Again we use Corollary 1, to find i_2, the index of 2 as the item on left of i_3, just smaller than $P[i_3]$. We use Corollary 2, to find i_4, the index of 4 as the rightmost item larger than $P[i_3]$. Finally, $i_1 = RangeMinimaP(i_3, i_4)$.

3412 Fix 4. 3 can be found as the largest element smaller than 4 on the left side of 4 using Corollary 1. We create an array of right near larger. For each element e in P that does not have another element f pointing to it

(f's right near larger be e) we will change the value of e to max and we will call this array R_1. We then use $RangeMinimaR_1(i_4, n)$ to find 2. 1 would be the closest element on the left of 2 that use 2 as its right near larger.

2413 This is reverse of pattern 3142. This is the test of separability of the text. Linear time algorithm is known for this case [5].

1342 Fix 3. 4 is right nearest larger. 1 can be found by $RangeMinimaP(1, i_3)$. Now in the *Position* array use $RangeMaximaPosition(P[i_1], P[i_3])$ to find the position, i.e. i_2 (i.e. use Corollary 2).

Theorem 3. Given any length 4 permutation (other than 1324 and its reverse 4231), we can test whether it is present (involved) in another permutation of length n in $\theta(n)$ time. □

4 Permutation 1324

In array P we use right nearest larger to build a forest. Within each tree of the forest we define the chain of the tree consisting of the root r of the tree, the largest child c_1 of r, the largest child c_2 of c_1, ..., the largest child c_{i+1} of c_i, ..., etc. This is illustrated in Fig. 1.

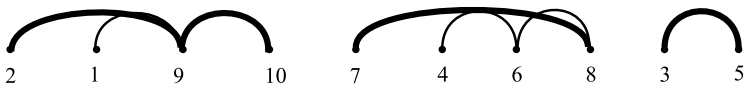


Fig. 1. Forest of right nearest larger. Chains are shown in boldface.

Our intension is to let roots of trees serve as 4 and nodes on the chains to serve as 3. 1 will be found using range minima and 2 will be served by non-chain nodes.

We traverse a tree this way: when we are at node d, we visit the subtrees of d in the order from the subtree rooted at the smallest child of d to the subtree rooted at the largest child of d. After that we visit d. We start at root of the tree. This traversal will label the nodes of the tree.

We start from the rightmost tree and visiting nodes in this tree in the order of the above traversal. Let d be a non-chain node we are visiting and let $d, d_1, d_2, ..., d_t$ be the path in the tree from d to the nearest ancestor d_t where d_t has another child c larger than d. In this case c will be larger than $d, d_1, d_2, ..., d_{t-1}$ and c will be on the left side of $d, d_1, ..., d_{t-1}, d_t$ in array P. Thus we can let d_{t-1} serve as 2, c serve as 3 and d_t serve as 4. 1 will be found using $RangeMinimaP(1, i_3)$. If $RangeMinimaP(1, i_3)$ is larger than 2, then we label $d, d_1, ..., d_{t-1}$ as they cannot serve as 2. They cannot serve as 1 or 3 because of our traversal order (i.e. they may have tried to serve as 1 or 3 before in the traversal).

Thus after we examined all non-chain nodes they cannot serve as 1, nor 2, nor 3. They need not serve as 4 because the root of the tree can serve as 4. Thus these non-chain nodes can be removed.

The chain nodes may later serve as 2 but they cannot serve as 1 or 3 because there are no qualifying 2 for them. All of them except the root need not serve as 4 because the root can serve 4 for them.

Next we examine the second rightmost tree. After we did the same examination for the tree as we did for the rightmost tree only the chain nodes remains to be tested as 2's. However, we have to test the chain nodes in the first tree as 2's using nodes in the second tree as 3's.

In order to do this we have the two lists of nodes each sorted in ascending order. One is the list L_1 of the chain nodes of the first tree and the second list L_2 is the one containing all the tree nodes of the second tree. Let r_1 be the root of the first tree and r_2 be the root of the second tree. We visit these two lists from smallest nodes.

Let a be the current smallest node in L_1 and b be the current smallest node in L_2. If $b < a$ then we remove b from further comparison and get next smallest node from L_2. In this case b is less than all remaining nodes in L_1 and therefore cannot serve as 3 for the remaining nodes in L_1 to serve as 2's. If $r_1 > b > a$ then we let r_1 serve as 4, b serve as 3 and a serve as 2 and use $RangeMinimaP(1, i_3)$ to find 1. If $RangeMinimaP(1, i_3) > 2$ then if a's chain parent p is less than b then p can replace a to serve as 2 and a can be deleted. If $p > b$ then b can be removed. In either case we remove one node. If $r_1 < b$ then we can stop because the remaining nodes cannot serve as 3 because there is no 4 for them.

Thus we visit nodes in the second tree at most twice. In the remaining we have the chain for the second tree left and some nodes on the chain for the first tree left. We can merge these nodes from two chains together and form an ascending list. The merging is not done by examining all nodes of the two chains as doing this way will be too costly. We maintain the above two ascending lists in linked lists. Thus once we find $r_1 < b$ for b as a chain node then we insert the remaining nodes in L_1 between b and b's chain child (here insert into the chain of the second tree and not inserting into L_2). Thus the merge takes constant time. Note now all b's chain descendants are smaller than the remaining nodes in L_1.

Then we view the merged chain node as one chain and we continue working on the third tree from right.

What we have described is the linear time algorithm for pattern 1324.

Theorem 4. Pattern 1324 can be tested whether it is present (involved) in another permutation of length n in $\theta(n)$ time. □

5 Concluding Remarks

We have described $\theta(n)$ time algorithms for testing involvement of all length 4 patterns. Previously most of these patterns have only $O(n \log n)$ time algorithms.

References

1. Albert, M.H., Aldred, R.E.L., Atkinson, M.D., Holton, D.A.: Algorithms for Pattern Involvement in Permutations. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 355–367. Springer, Heidelberg (2001)
2. Berkman, O., Matias, Y., Ragde, P.: Triply-logarithmic parallel upper and lower bounds for minimum and range minima over small domains. Journal of Algorithms 28, 197–215 (1998)
3. Berkman, O., Schieber, B., Vishkin, U.: Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. Journal of Algorithms 14, 344–370 (1993)
4. Bose, P., Buss, J.F., Lubiw, A.: Pattern matching for permutations. Information Processing Letters 65, 277–283 (1998)
5. Fredman, M.L.: On Computing the length of longest increasing subsequences. Discrete Mathematics 11, 29–35 (1975)
6. Ibarra, L.: Finding pattern matchings for permutations. Information Processing Letters 61, 293–295 (1997)
7. Kitaev, S., Mansour, T.: A survey on certain pattern problems, `http://www.ru.is/kennarar/sergey/index_files/Papers/survey.ps`, `http://ajuarna.staff.gunadarma.ac.id/Downloads/files/1662/survey.pdf`
8. Preparata, F.P., Shamos, M.I.: Computational Geometry, An Introduction. Springer (1985)
9. Saxena, S.: Dominance made simple. Information Processing Letters 109, 419–421 (2009)
10. Tamassia, R., Vitter, J.S.: Optimal cooperative search in fractional cascaded data structures. Algorithmica 15, 154–171 (1996)
11. Yugandhar, V., Saxena, S.: Parallel algorithms for separable permutations. Discrete Applied Mathematics 146, 343–364 (2005)

Partial Degree Bounded Edge Packing Problem with Arbitrary Bounds

Pawan Aurora, Sumit Singh, and Shashank K. Mehta

Indian Institute of Technology, Kanpur - 208016, India
{paurora,ssumit}@iitk.ac.in, skmehta@cse.iitk.ac.in

Abstract. Given a graph $G = (V, E)$ and a non-negative integer c_u for each $u \in V$. Partial Degree Bounded Edge Packing (PDBEP) problem is to find a subgraph $G' = (V, E')$ with maximum $|E'|$ such that for each edge $(u, v) \in E'$, either $deg_{G'}(u) \leq c_u$ or $deg_{G'}(v) \leq c_v$. The problem has been shown to be NP-hard even for uniform degree constraint (i.e., all c_u being equal). Approximation algorithms for uniform degree cases with c_u equal to 1 and 2 with approximation ratio of 2 and 32/11 respectively are known. In this work we study general degree constraint case (arbitrary degree constraint for each vertex) and present two combinatorial approximation algorithms with approximation factors 4 and 2. We also study a related integer program for which we present an iterative rounding algorithm with approximation factor $1.5/(1-\epsilon)$ for any positive ϵ. This also leads to a $3/(1-\epsilon)^2$ factor approximation algorithm for the general PDBEP problem. For special cases (large values of $c_v/deg_G(v)$'s) the factor improves up to $1.5/(1-\epsilon)$. Next we study the same problem with weighted edges. In this case we present a $2 + \log_2 n$ approximation algorithm. In the literature exact $O(n^2)$ complexity algorithm for trees is known in case of uniform degree constraint. We improve this result by giving an $O(n \cdot \log n)$ complexity exact algorithm for trees with general degree constraint.

Keywords: Edge-Packing Problems, Iterative Rounding, Lagrangian Relaxation.

1 Introduction

The *partial degree bounded edge packing problem* (PDBEP) is described as follows: Given a graph $G = (V, E)$ and degree-bound function $c : V \to \mathbb{Z}^*$ ($\mathbb{Z}^*$ is the set of non-negative integers), compute a maximum cardinality set $E' \subseteq E$ which satisfies the *degree condition*: $(d'_u \leq c_u) \vee (d'_v \leq c_v)$ for each $e = (u, v) \in E'$. Here d'_x denotes the degree of vertex x in the graph $G' = (V, E')$. Without loss of generality, we will assume that $c_v \leq d_v$ for all $v \in V$ where d_v denotes the degree of v in G.

In the weighted version of the problem edges are assigned non-negative weights and we want to compute a set of edges E' with maximum cumulative weight subject to the degree condition described above.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 24–35, 2013.

In [4], the PDBEP problem was motivated by an application in binary string representation. It was shown there that the maximum expressible independent subset (MEIS) problem on 2-regular set can be reduced to PDBEP problem with uniform constraint $c = 2$. The PDBEP problem finds another interesting application in resource allocation. Given n types of resources and m jobs, each job needs two types of resources. A job j, which requires resources u and v, can be accomplished if u is not shared by more than c_u jobs or v is not shared by more than c_v jobs. Interpreting the resources as the vertices of the input graph and the jobs as edges, the PDBEP problem is to compute the maximum number of jobs that can be accomplished.

1.1 Related Work

The decision problem of edge packing when there is a uniform degree constraint of 1 is a parametric dual of the Dominating Set (DS) problem. It was studied in [3]. It was also studied under the framework of parameterized complexity by Dehne, Fellows, Fernau, Prieto and Rosamond in [1].

Recently Peng Zhang [4] showed that the PDBEP problem with uniform degree constraint ($c_v = k$ for all v) is NP-hard even for $k = 1$ for general graphs. They gave approximation algorithms for the PDBEP problem under uniform degree constraints of $k = 1$ and $k = 2$ with approximation factors 2 and 32/11 respectively. They showed that PDBEP on trees with uniform degree constraint can be solved exactly in $O(n^2)$ time.

1.2 Our Contribution

We propose three different approximation algorithms for the problem with general degree constraints (i.e., for arbitrary non-negative function c). Two of these algorithms are combinatorial in nature and their approximation ratios are 4 and 2. The third algorithm is a consequence of studying a related integer program (IP) for which we present a $1.5/(1-\epsilon)$ approximation iterative rounding [2] algorithm. It turns out that any α approximation of this IP is a $2\alpha/(1-\epsilon)$ approximation of the PDBEP problem for any $\epsilon > 0$. That gives us a $3/(1-\epsilon)^2$ factor approximation to the PDBEP problem. However for large degree constraint with respect to the degree, the approximation factor can improve up to $1.5/(1-\epsilon)$. The results detailed above are for general graphs with arbitrary degree constraint and the approximation factor is also improved for $c_v = 2$ (uniform constraint) case in [4].

Next we consider the PDBEP problem with arbitrary degree constraint for edge-weighted graphs. In this case we present a combinatorial approximation algorithm with approximation factor of $2+\log_2 n$. Edge-weighted PDBEP problem is not addressed in the literature, to the best of our knowledge.

Finally we present an exact algorithm for unweighted trees with arbitrary degree constraint function. The time complexity of this algorithm is $O(n \log n)$. This is an improvement over the $O(n^2)$ algorithm in [4] which is applicable to only the uniform degree constraint case.

2 Approximation Algorithms for the Unweighted Case

The optimum solution of a PDBEP problem can be bounded as follows.

Lemma 1. *Let $G = (V, E)$ be a graph with degree-bound function $c : V \to \mathbb{Z}^*$. Then the optimal solution of PDBEP can have at most $\sum_{v \in V} c_v$ edges.*

Proof. Let $E' \subset E$ be a solution of PDBEP. Let $U = \{v \in V | d'_v \leq c_v\}$. Then from the degree condition we see that U is a vertex cover in the graph (V, E'). Hence $|E'| \leq \sum_{u \in U} c_u \leq \sum_{v \in V} c_v$. □

2.1 Edge Addition Based Algorithm

Compute a maximal solution Y by iteratively adding edges, i.e., in each iteration select a new edge and add it to Y if it does not result into degree violation on both end-vertices. Let $d_Y(x)$ denote the degree of a vertex x in the graph (V, Y). Partition the vertex set into sets: $A = \{v | d_Y(v) < c_v\}$, $B = \{v | d_Y(v) = c_v\}$, and $C = \{v | d_Y(v) > c_v\}$. Observe that every edge of the set $E \setminus Y$ which is incident on a vertex in A, has its other vertex in B. Hence for any $a_1, a_2 \in A$ the $E \setminus E'$ edges incident on a_1 are all distinct from those incident on a_2. Construct another edge set Z containing any $c_v - d_Y(v)$ edges from $E \setminus Y$, incident on v for each $v \in A$. Observe that Z also satisfies the degree constraints. Output the larger of Y and Z. See Algorithm 1. We have the following result about the correctness of the algorithm.

Lemma 2. *The algorithm outputs a set which satisfies the degree constraint.*

Consider the set $Y \cup Z$. In this set the degree of each vertex is not less than its degree-bound. Hence the cardinality of the output of the algorithm is at least $\sum_v c_v/4$. From Lemma 1 the approximation ratio is bounded by 4.

Theorem 1. *The Algorithm 1 has approximation factor* 4.

2.2 Edge Deletion Based Algorithm

The second algorithm for PDBEP is based on elimination of edges from the edge set. Starting with the input edge set E, iteratively we delete the edges in violation, i.e., in each iteration one edge (u, v) is deleted if the current degree of u is greater than c_u and that of v is greater than c_v. The surviving edge set Y is the result of the algorithm. See Algorithm 2.

Clearly Y satisfies the degree condition. Also observe that $d_Y(v) \geq c_v$ for all $v \in V$. Hence $|Y| \geq \sum_v c_v/2$. From Lemma 1, $|Y| \geq OPT/2$ where OPT denotes the optimum solution.

Theorem 2. *The Algorithm 2 has approximation ratio* 2.

Data: A connected graph $G = (V, E)$ and a function $c : V \to \mathbb{Z}^*$ such that $c_v \leq d(v)$ for each vertex v.
Result: Approximation for the largest subset of E which satisfies the degree-condition.
$Y := \emptyset$;
for $e \in E$ **do**
  if $Y \cup \{e\}$ *satisfies the degree-condition* **then**
    $Y := Y \cup \{e\}$;
  end
end
Compute $A := \{v \in V | d_Y(v) < c_v\}$;
$Z := \emptyset$;
for $v \in A$ **do**
  Select arbitrary $c_v - d_Y(v)$ edges incident on v in $E \setminus Y$ and insert into Z;
end
if $|Y| \geq |Z|$ **then**
  return Y;
else
  return Z;
end

Algorithm 1. Edge Addition Based Algorithm

2.3 LP Based Algorithm

In this section we explore a linear programming based approach to design an approximation algorithm for PDBEP.

The Integer Program. The natural IP formulation of the problem is as follows.

$$\text{IP1: max } \psi = \sum_{e \in E} y_e, \text{ subject to}$$
$$y_e \leq x_u + x_v \; \forall e = (u, v) \in E,$$
$$\sum_{e \in \delta(v)} y_e \leq c_v x_v + d_v(1 - x_v) \; \forall v \in V,$$
$$\text{where } \delta(v) \text{ denotes the set of edges incident on } v,$$
$$x_v \in \{0, 1\} \; \forall v \in V, \; y_e \in \{0, 1\} \; \forall e \in E$$

The solution computed by the program is $E' = \{e | y_e = 1\}$. The linear programming relaxation of the above integer program will be referred to as LP1.

Lemma 3. *The integrality gap of LP1 is $\Omega(n)$ where n is the number of vertices in the graph.*

Proof. Consider the following instance of the problem. Let G be a complete graph on n vertices $\{v_0, v_1, \ldots, v_{n-1}\}$ and the degree constraint be $c_v = 1 \, \forall v \in V$. We now construct a feasible fractional solution of $LP1$ as follows. Let $x_v = 0.5$ for all

Data: A connected graph $G = (V, E)$ and a function $c : V \to \mathbb{Z}^*$ such that c_v is the degree bound for vertex v.
Result: Approximation for the largest subset of E which satisfies the degree-condition.
$Y := E$;
for $e = (u, v) \in Y$ **do**
  if $d_Y(u) > c_u$ *and* $d_Y(v) > c_v$ **then**
    $Y \leftarrow Y \setminus \{e\}$;
  end
end
return Y;

Algorithm 2. Edge Deletion Based Algorithm

v and $y_e = 1$ for all $e = (v_i, v_j)$ where j is in the interval $((i - \lfloor n/4 \rfloor) (\mathrm{mod}\ n), (i + \lfloor n/4 \rfloor) (\mathrm{mod}\ n))$. The value of the objective function for this solution is at least $(n-1)^2/4$. On the other hand, from Lemma 1, the optimal solution for the IP1 cannot be more than n. Hence the integrality gap is $\Omega(n)$. □

High integrality gap necessitates an alternative approach.

Approximate Integer Program. We propose an alternative integer program IP2, for any $\epsilon > 0$, which is a form of Lagrangian relaxation of IP1. We will show that its *maximal* solutions are also solutions of IP1 and any α approximation of IP2 is a $2\alpha/(1-\epsilon)$ approximation of IP1. A maximal solution of IP2 is a solution in which changing the value of any y_e or any z_v either renders the solution infeasible or does not improve its objective function value. It is easy to show that in a maximal solution $z_v = \max\{0, \sum_{e \in \delta(v)} y_e - c_v\}$ for all v.

$$\text{IP2: max } \phi = 2 \sum_{e \in E} y_e - (1 + \epsilon) \sum_{v \in V} z_v, \text{ subject to}$$
$$\sum_{e \in \delta(v)} y_e \leq c_v + z_v \ \forall v \in V,$$
$$z_v \in \{0, 1, 2, \ldots\} \ \forall v \in V,\ y_e \in \{0, 1\} \ \forall e \in E$$

Note that any subset of edges E' is a feasible solution of IP2 if we choose $z_v = \max\{0, \sum_{e \in \delta(v)} y_e - c_v\}$ for all v. Besides, these values of z will give maximum value of the objective function over other consistent values of z. Hence z values are not required to be specified in the solutions of IP2. We will denote $\sum_v z_v$ by Z.

Lemma 4. *Every maximal solution of the integer program IP2 is also a feasible solution of PDBEP.*

Proof. Consider any maximal solution E' of IP2. In a maximal solution $z_v = \max\{0, \sum_{e \in \delta(v)} y_e - c_v\}$ for all v. Assume that it is not a feasible solution of PDBEP. Then there must exist an edge $e = (u, v) \in E'$ such that $z_u \geq 1$ and

$z_v \geq 1$. Define an alternative solution $E'' = E' \setminus \{e\}$ and decrement z_u and z_v by 1 each. Observe that the objective function of the new solution increases by 2ϵ. This contradicts that E' is a maximal solution. □

In a maximal solution E' of IP2, Z is the count of excess edges incident on violating vertices. Hence $Z \leq |E'|$. In this case it is easy to see that $Z/|E'| \leq \max_v\{(d_v - c_v)/d_v\}$.

Lemma 5. *Any α approximate solution of IP2, which is also maximal, is a $2\alpha/(1-\epsilon)$ approximation of PDBEP problem.*

Proof. Let E'_1 be an α-approximation maximal solution of IP2 and E'_2 be an optimum solution of PDBEP. Then E'_2 is also a solution of IP2 with $y_{2e} = 1$ for $e \in E'_2$ and $z_{2v} = \max\{0, \sum_{e\in\delta(v)} y_{2e} - c_v\}$. Then $\phi(E'_1) = 2|E'_1| - (1+\epsilon)Z_1$ and $\phi(E'_2) = 2|E'_2| - (1+\epsilon)Z_2$. Let OPT denote the optimum value of IP2. Then $\phi(E'_2) \leq OPT$ and $OPT/\alpha \leq \phi(E'_1)$. So $2|E'_2| - (1+\epsilon)Z_2 \leq \alpha(2|E'_1| - (1+\epsilon)Z_1)$. Suppose $Z_2 \leq \beta|E'_2|$. Then $|E'_2|/|E'_1| \leq 2\alpha/(2-(1+\epsilon)\beta)$. Since $\beta \leq 1$, we get the desired approximation factor. □

Corollary 1. *If $c_v \geq (1-\beta)d_v \, \forall v$, then any α approximate solution of IP2, which is also maximal, is a $2\alpha/(2-(1+\epsilon)\beta)$ approximation of PDBEP.*

In this case $Z_2 \leq \beta|E'_2|$. Now the result follows from the previous proof.

2.4 Algorithm for IP2

We propose Algorithm 3 which approximates the IP2 problem within a constant factor of approximation. LP2 is the linear program relaxation of IP2. Here we assume that an additional constraint is imposed, namely, $\{z_v = 0 | v \in C\}$ where we require a solution in which every $v \in C$ must necessarily satisfy the degree constraint. The input to the problem is $(H = (V, E), C)$. Algorithm starts with $E' = \emptyset$ and builds it up one edge at a time by iterative rounding. In each iteration we discard at least one edge from further consideration. Hence it requires at most $|E|$ iterations (actually it requires at most $|V|+1$ iterations, see the remark below.) To simplify the analysis Algorithm 3 is presented in the recursive format.

$$\text{LP2: max } \phi = 2\sum_{e\in E} y_e - (1+\epsilon)\sum_{v\in V} z_v, \text{ subject to}$$

$$\sum_{e\in\delta(v)} y_e \leq c_v + z_v \; \forall v \in V \setminus C,$$

$$\sum_{e\in\delta(v)} y_e \leq c_v \; \forall v \in C,$$

$$z_v \geq 0 \; \forall v \in V, \; y_e \geq 0 \; \forall e \in E, \; -y_e \geq -1 \; \forall e \in E$$

In the following analysis we will focus on two problems: (H, C) of some i-th nested recursive call and (H_1, C_1) of the next call in Algorithm 3. For simplicity we will refer to them as the problems of graphs H and H_1 respectively.

Lemma 6. *In a corner solution of LP2 on a non-empty graph there is at least one edge e with $y_e = 0$ or $y_e \geq 1/2$.*

Proof. Assume the contrary that in an extreme point solution of LP2 all y_e are in the open interval $(0, 1/2)$. Let us partition the vertices as follows. Let n_1 vertices have $c_v > 0$ and $z_v > 0$, n_2 vertices have $c_v > 0$ and $z_v = 0$ and n_3 vertices have $c_v = 0$ and $z_v > 0$. Note that the case of $c_v = 0$ and $z_v = 0$ cannot arise because $y_e > 0$ for all e. In each case let n_i' vertices have the condition $\sum_{e \in \delta(v)} y_e \leq c_v + z_v$ tight (an equality) and n_i'' vertices have the condition a strict inequality. Let the number of edges be m.

The total number of variables is $n_1+n_2+n_3+m$. In $n_1'+n_2'$ cases $\sum_{e \in \delta(v)} y_e = c_v + z_v$ where $c_v \geq 1$ and each $y_e < 0.5$ so there must be at least 3 edges incident on such vertices. Since the graph has no isolated vertices, every vertex has at least one incident edge. Hence $m \geq (3n_1' + 3n_2' + n_1'' + n_2'' + n_3)/2$. So the number of variables is at least $n_1' + n_2' + (1.5)(n_1 + n_2 + n_3)$.

Now we find the number of tight conditions. None of the y_e touches their bounds. The number of z_v which are equal to zero is n_2, and the number of instances when $\sum_{e \in \delta(v)} y_e = c_v + z_v$ is $n_1' + n_2' + n_3'$. Hence the total number of conditions which are tight is $n_2 + n_1' + n_2' + n_3'$. Since the solution is an extreme point, the number of tight conditions must not be less than the number of variables. So $n_2 + n_1' + n_2' + n_3' \geq n_1' + n_2' + (1.5)(n_1 + n_2 + n_3)$. This implies that $n_1 = n_2 = n_3 = 0$, which is absurd since the input graph is not empty. □

Remark: The program LP2 has $|E| + |V|$ variables and $2|E| + 2|V|$ constraints. Hence in the first iteration the optimal solution must have at least $|E| - |V|$ tight edge-constraints (i.e., $y_e = 0$ or $y_e = 1$.) All these can be processed simultaneously so in the second round at most $|V|$ edges will remain in the residual graph. Thus the total number of iterations cannot exceed $|V| + 1$.

Lemma 7. *If $y_e \geq 1/2$ in the solution of LP2 where $e = (u, v)$, then $(c_u > 0, z_u = 0)$ or $(c_v > 0, z_v = 0)$.*

Proof. Assume that $z_v > 0$ and $z_u > 0$ in the solution. Let the minimum of z_v, z_u, and y_e be β. Subtracting β from these variables results in a feasible solution with objective function value greater than the optimum by $2\beta \cdot \epsilon$. This is absurd. Hence z_u and z_v both cannot be positive.

Next assume that $c_u = 0$ and $z_u = 0$. Then y_e must be zero, contradicting the fact that $y_e \geq 1/2$. Similarly $c_v = 0$ and $z_v = 0$ is also not possible.

Therefore at least one of $(c_u > 0, z_u = 0)$ and $(c_v > 0, z_v = 0)$ holds. □

Lemma 8. *The Algorithm 3 returns a feasible solution of PDBEP.*

Proof. The claim is trivially true when the graph is empty. We will use induction.

In the case of $y_e = 0$, the solution of H_1 is also a solution of H. From induction hypothesis it is feasible for PDBEP for H_1 hence it is also feasible for PDBEP for H.

Consider the second case, i.e., $y_e \geq 1/2$. Let $e = (u, v)$. From Lemma 7 $f_v = a > 0$ and $z_v = 0$. Since $z_{1v} = 0$ and $f_{1v} = a - 1$, in the solution of H_1 at

most $a-1$ edges can be incident on v. So in the solution of H, there are at most a edges incident on v and $f_v = a$. Thus e is valid in the solution of H.

Now consider vertex u in this case. If $f_u > 0$, then it is similar to v. But if $f_u = 0$, then f_{1u} is also 0. If $z_{1u} > 0$, i.e., the solution of H_1 has u violating, then edges incident on u must have their other ends on non-violating vertices. Replacing e will not affect any other edges. Next if $z_{1u} = 0$ then solution of H_1 will have no edge incident on u. Now putting back e, we find e is incident on u and $f_u = 0$ so u turns into a violating vertex ($z_u = 1$). But e is the only edge incident on u and its other end v is not violating. Other edges are valid due to induction hypothesis. Hence the solution of H is a feasible solution of PDBEP. □

Now we analyze the performance of the algorithm.

Data: A connected graph $G = (V, E)$ and a function $c : V \to \mathbb{Z}^*$
Result: A solution of PDBEP problem.
for $v \in V$ **do**
  $f_v := c_v$;
end
$C := \emptyset$;
$E' := SolveIP2(G, C, f)$; `/* see the function SolveIP2 */`
return E';

Algorithm 3. Iterative Rounding based Algorithm in Recursive Format

Lemma 9. *Algorithm 3 gives a* $1.5/(1-\epsilon)$ *approximation of IP2.*

Proof. Let c denote $1.5/(1-\epsilon)$. We will denote the optimal LP2 solutions of H and H_1 by F and F_1 respectively. Similarly I and I_1 will denote the solutions computed by the algorithm for H and H_1 respectively. f_{1*} and z_{1*} denote the parameters associated with H_1. We will assume that $z_x = \max\{0, \sum_{e\in\delta(x)} y_e - f_x\}$ for integral solutions to compute their ϕ-values. Again we will prove the claim by induction. The base case is trivially true. From induction hypothesis $\phi(F_1)/\phi(I_1) \leq c$ and our goal is to show the same bound holds for $\phi(F)/\phi(I)$.

In the event of $y_e = 0$ in F, $\phi(F) = \phi(F_1)$ and $\phi(I) = \phi(I_1)$. Hence $\phi(F)/\phi(I) = \phi(F_1)/\phi(I_1)$.

In case $y_e = \alpha \geq 1/2$ we will consider two cases: (i) $f_u > 0$ and (ii) $f_u = 0$ in F. In the first case I differs from I_1 in three aspects: $y_e = 1$ in I, $f_v = f_{1v} + 1$ and $f_u = f_{1u} + 1$. So z_v and z_u remain unchanged, i.e., $z_v = z_{1v}$ and $z_u = z_{1u}$. Thus $\phi(I) = \phi(I_1) + 2$. In the second case also y_e increases by 1 and z_v remains unchanged but z_u increases by 1 because in this case $f_u = f_{1u} = 0$. Hence $\phi(I) = \phi(I_1) + 1 - \epsilon$.

In the remaining part of the proof we will construct a solution of LP2 for H_1 from F, the optimal solution of LP2 for H.

Again we will consider the two cases separately. First the case of $f_u > 0$. Set $y_e = 0$. If $\sum_{e'\in\delta(v)\setminus\{e\}} y_{e'} \geq 1 - \alpha$ then subtract the values of $y_{e'}$ for $e' \in \delta(v) \setminus \{e\}$ in arbitrary manner so that the sum $\sum_{e'\in\delta(v)\setminus\{e\}} y_{e'}$ decreases by $1-\alpha$. If $\sum_{e'\in\delta(v)\setminus\{e\}} y_{e'} < 1-\alpha$, then set $y_{e'}$ to zero for all edges incident on v.

Function: SolveIP2($H = (V_H, E_H), C, f$)
if $E_H := \emptyset$ **then**
  return $\emptyset$;
end
Delete all isolated vertices from V_H;
$(\mathbf{y}, \mathbf{z}) = \text{LPSolver}(H, C)$;
`/* solve LP2 with degree-bounds` $f(x)$ `for all` $x \in V_H$ `*/`
if $\exists e \in E_H$ *with* $y_e = 0$ **then**
  $H_1 := (V_H, E_H \setminus \{e\})$;
  $C_1 := C$;
  $E' := SolveIP2(H_1, C_1, f)$;
else
  From Lemma 6 there exists an edge $e := (u, v)$ with $y_e \geq 1/2$;
  From Lemma 7 without loss of generality we assume $(f_v > 0, z_v = 0)$;
  $f_v := f_v - 1$;
  $C_1 := C \cup \{v\}$;
  $f_u := \max\{f_u - 1, 0\}$;
  $H_1 := (V_H, E_H \setminus \{e\})$;
  $E' := SolveIP2(H_1, C_1, f) \cup \{e\}$;
  `/* Including` e `in` E' `means` y_e `is rounded up to 1. In case` $f_u = 0$`,`
    z_u `is implicitly raised to ensure that` $\sum_{e' \in \delta(u)} y_{e'} \leq f_u + z_u$
    `continues to hold. We do not explicitly increase` z_u `value`
    `since it is not output as a part of the solution.` `*/`
end
return E';

Repeat this step for edges incident on u. Retain values of all other variables as in F (in particular, the values of z_u and z_v). Observe that these values constitute a solution of LP2 for H_1. Call this solution F_1'. From induction hypothesis $\phi(F_1) \leq c\phi(I_1)$. Hence we have $\phi(F_1') \leq c\phi(I_1)$. Then $\phi(F_1') \geq \phi(F) - 2(1 + 1 - \alpha) \geq \phi(F) - 3$. We have $\phi(F) \leq \phi(F_1') + 3 \leq c\phi(I_1) + 3 = c(\phi(I) - 2) + 3 \leq c\phi(I)$.

In the second case $f_u = 0$. Once again repeat the step described for edges incident on v and set y_e to zero. In this case $z_u \geq \alpha$ so subtract α from it. It is easy to see that again the resulting variable values form an LP2 solution of H_1, call it F_1'. So $\phi(F_1') = \phi(F) - (2 - (1 + \epsilon)\alpha)$. So $\phi(F) = \phi(F_1') + (2 - (1 + \epsilon)\alpha) \leq c\phi(I_1) + (2 - (1 + \epsilon)\alpha)$. Plugging $\phi(I) - 1 + \epsilon$ for $\phi(I_1)$ and simplifying the expression gives $\phi(F) \leq c\phi(I)$. This completes the proof. □

Combining lemmas 5 and 9 we have the following result.

Theorem 3. *Algorithm 3 approximates PDBEP with approximation factor* $3/(1 - \epsilon)^2$.

From Corollary 1 we have the following result.

Corollary 2. *If* $c_v \geq (1 - \beta)d_v$ *for all* v*, then Algorithm 3 approximates PDBEP with approximation factor* $3/((2 - (1 + \epsilon)\beta)(1 - \epsilon))$.

3 Approximation Algorithm for the Weighted Case

Let $H(v)$ denote the heaviest c_v edges incident on vertex v, called *heavy set* of vertex v. Then from a generalization of Lemma 1 the optimum solution of PDBEP in weighted-edge case is bounded by $\sum_{v \in V} \sum_{e \in H(v)} w(e)$ where $w(e)$ denotes the weight of edge e. We will describe a method to construct upto $1 + \log_2 |V|$ solutions, which cover $\cup_{v \in V} H(v)$. Then the heaviest solution gives a $2 + \log_2 |V|$ approximation of the problem.

3.1 The Algorithm

Input: A graph (V, E) with non-negative edge-weight function $w()$. Let $|V| = n$.

Step 0: Add infinitesimally small weights to ensure that all weights are distinct, without affecting heavy sets.

Step 1: $E_1 = E \setminus \{e = (u, v) \in E | e \notin H(u) \text{ and } e \notin H(v)\}$.

Step 2: $T = \{e = (u, v) \in E | e \in H(u) \text{ and } e \in H(v)\}$.

Step 3: $E_2 = E_1 \setminus T$. Clearly each edge of E_2 is in the heavy set of only one of its end-vertices. Suppose $e = (u, v) \in E_2$ with $e \notin H(u)$ and $e \in H(v)$. Then we will think of e as directed from u to v.

Step 4: Label the vertices from 0 to $n - 1$ such that if edge (u, v) is directed from u to v, then $Label(u) < Label(v)$. Define subsets of E_2-edges, $A_0, \ldots, A_{k-1}$, where $k = \log_2 n$, as follows. A_r consists of edges (u, v) directed from u to v, such that the most significant $r - 1$ bits of binary expansion of the labels of u and v are same and r-th bit differs. Note that this bit will be 0 for u.

Step 5: Output that set among the $\log_2 n + 1$ sets, $T, A_0, \ldots, A_{k-1}$, which has maximum cumulative edge weight.

Theorem 4. *The algorithm gives a feasible solution with approximation factor* $2 + \log_2 n$.

Proof. Set T constitutes a feasible solution since both ends of each edge in it satisfy the degree constraint. The directed E_2 edges define an acyclic graph, hence the labeling can be performed by topological sorting. Clearly $E_2 = \cup_r A_r$. In A_r all arrows are pointed from u with r-th most significant bit zero to v with r-th most significant bit one. Hence it is a bipartite graph where all arrows have heads in one set and the tails in the other. All vertices on the head side satisfy the degree conditions because all their incident edges are in their heavy sets. Therefore A_r are feasible solutions. We have $T \cup (\cup_r A_r) = E_1$. Observe that $\cup_v H(v) = E_1$. Only T-edges have both ends in heavy sets. Using the fact that $OPT \leq \sum_v w(H(v))$, we deduce that $OPT \leq 2w(T) + \sum_r w(A_r)$. So the weight of the set output in step 5 is at least $OPT/(2 + \log_2 n)$. □

4 Exact Algorithm for Trees

In this section we give a polynomial time exact algorithm for the unweighted PDBEP problem for the special case when the input graph is a tree. We will

denote the degree of a vertex v in the input graph by $d(v)$ and its degree in a solution under consideration by $d'(v)$.

Let T be a rooted tree with root R. For any vertex v we denote the subtree rooted at v by $T(v)$. Consider all feasible solutions of PDBEP of graph $T(v)$ in which degree of v is at most $c_v - 1$, call them H-solutions (white). Let $h(v)$ be the number of edges in the largest such solution. Similarly let $g(v)$ be the optimal G-solution (grey) in which the degree of v is restricted to be equal to c_v. Lastly $b(v)$ will denote the optimal B-solution (black) which are solutions of $T(v)$ under the restriction that degree of v be at least c_v and every neighbor of v in the solution satisfies the degree condition. It may be observed that one class of solutions of $T(v)$ are included in G-solutions as well as in B-solutions. These are the solutions in which $d'(v) = c_v$ and every neighbor u of v in the solution has $d'(u) \leq c_u$. If in any of these cases there are no feasible solutions, then the corresponding optimal value is assumed to be zero. Hence the optimum solution of PDBEP for T is the maximum of $h(R), g(R)$, and $b(R)$ and all three values are zero for leaf nodes.

Let $Ch(v)$ denote the set of child-nodes of v in $T(v)$. We partition $Ch(v)$ into $H(v) = \{u \in Ch(v) | h(u) \geq \max\{g(u), b(u)\}\}$, $G(v) = \{u \in Ch(v) | g(u) > \max\{h(u), b(u)\}\}$, $B(v) = Ch(V) \setminus (G(v) \cup H(v))$. While constructing a G-solution of $T(v)$ from the solutions of the children of v we can include the edge (v, u) for any vertex u in $H(v) \cup B(v)$ along with the optimal solution of $T(u)$ without disturbing the degree conditions of the edges in this solution. But we can add edge (v, u) to the solution, for any $u \in G(v)$, only by selecting a B-solution or an H-solution of $T(u)$ because if we use a G-solution for $T(u)$, then vertex u which was earlier satisfying the degree condition, will now have degree $c_u + 1$.

Next, while constructing a B-solution of $T(v)$ we can connect v to any number of $H(v)$ vertices and use their optimal H-solutions. In the same case, in order to connect v with $u \in B(v) \cup G(v)$ we must use the optimal H-solution of $T(u)$ (which is not the best solution of $T(u)$).

Suppose we want to build the optimum G-solution of v. If $|H(v)| + |B(v)|$ is less than c_v, then we must pick additional $c_v - |H(v)| - |B(v)|$ vertices from $G(v)$ to connect with v. If $k = c_v - |H(v)| - |B(v)| > 0$, then we define $S'(v)$ to be the set of k members of $G(v)$ having least $g(u) - \max\{h(u), b(u)\}$. Otherwise $S'(v) = \emptyset$. $S'(v)$ are those vertices which we will like to connect v with.

Suppose we want to build the best B-solution for $T(v)$ and $|H(v)| < c_v$. Then we will connect v with exactly c_v children because connecting with any additional child will either keep the value same or make it worse because for connecting with a $G(v)$ or $B(v)$ node we will be forced to use their H-solution which is not their best solution. If $k = c_v - |H(v)| > 0$, then we define $S''(v)$ to be the set of k members of $G(v) \cup B(v)$ with smallest key values where key is $g(u) - h(u)$ for $u \in G(v)$ and $b(u) - h(u)$ for $u \in B(v)$. Otherwise $S''(v) = \emptyset$. Now we have following lemma which leads to a simple dynamic program for PDBEP.

Lemma 10. *For any internal vertex v of T,*

(i) $h(v) = \sum_{u \in B(v)} b(u) + \sum_{u \in H(v)} h(u) + \sum_{u \in G(v)} g(u) + \min\{c_v - 1, |H(v)| + |B(v)|\}$.
If $d(v) = c_v$ *and* $v \neq R$, *then set* $b(v) = g(v) = 0$ *otherwise*
(ii) $g(v) = \sum_{u \in B(v)} b(u) + \sum_{u \in H(v)} h(u) + \sum_{u \in G(v) \setminus S'(v)} g(u) + \sum_{u \in S'(v)} \max\{h(u), b(u)\} + c_v$.
(iii) $b(v) = \sum_{u \in H(v) \cup S''(v)} h(u) + \sum_{u \in B(v) \setminus S''(v)} b(u) + \sum_{u \in G(v) \setminus S''(v)} g(u) + \max\{c_v, |H(v)|\}$.

Observe that if $h(u)$ is equal to $b(u)$ or $g(u)$, then u is categorized as an $H(v)$ vertex and if $b(u) = g(u) > h(u)$, then u is assigned to $B(v)$ set. Hence the last term is maximum in each of the cases in the lemma.

The algorithm initializes $h(v), b(v)$, and $g(v)$ to zero for the leaf vertices and computes these values for the internal vertices bottom up. Finally it outputs the maximum of the three values of the root R. In order to compute $S'()$ and $S''()$ sets for each vertex, we need to sort the child nodes with respect to the key values. Thus at each vertex we incur $O(|Ch| \log |Ch|)$ cost, where Ch denotes the set of children of that vertex. Besides, ordering the vertices so that child occurs before the parent (topological sort) takes $O(n)$ time. Hence the time complexity is $O(n \log n)$.

Acknowledgement. We thank the referees of the paper for detailed feedback and suggestions which improved the analysis of the weighted case and also the overall presentation of the paper.

References

1. Dehne, F., Fellows, M., Fernau, H., Prieto, E., Rosamond, F.: NONBLOCKER: Parameterized algorithmics for MINIMUM DOMINATING SET. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 237–245. Springer, Heidelberg (2006)
2. Jain, K.: A factor 2 approximation algorithm for the generalized steiner network problem. Combinatorica 21(1), 39–60 (2001)
3. Nieminen, J.: Two bounds for the domination number of a graph. Journal of the Institute of Mathematics and its Applications 14, 183–187 (1974)
4. Zhang, P.: Partial degree bounded edge packing problem. In: Snoeyink, J., Lu, P., Su, K., Wang, L. (eds.) FAW-AAIM 2012. LNCS, vol. 7285, pp. 359–367. Springer, Heidelberg (2012)

Faster Exact Computation of rSPR Distance

Zhi-Zhong Chen[1] and Lusheng Wang[2]

[1] Department of Information System Design, Tokyo Denki University, Hatoyama, Saitama 350-0394, Japan
zzchen@mail.dendai.ac.jp
[2] Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong SAR
lwang@cs.cityu.edu.hk

Abstract. Due to hybridization events in evolution, studying two different genes of a set of species may yield two related but different phylogenetic trees for the set of species. In this case, we want to measure the dissimilarity of the two trees. The rooted subtree prune and regraft (rSPR) distance of the two trees has been used for this purpose, and many algorithms and software tools have been developed for computing the rSPR distance of two given phylogenetic trees. The previously fastest exact algorithm for this problem runs in $O\left(2.415^d n\right)$ time, where n and d are the number of leaves and the rSPR distance of the input trees, respectively. In this paper, we present a faster exact algorithm which runs in $O\left(2.344^d n\right)$ time. Our experiments show that the new algorithm can be significantly faster than the newest version (namely, v1.1.1) of the previously best software (namely, Whidden *et al.*'s *RSPR*) for rSPR distance.

Keywords: Phylogenetic tree, rSPR distance, fixed-parameter algorithm.

1 Introduction

When studying the evolutionary history of a set of existing species, one can obtain a phylogenetic tree of the set of species with high confidence by looking at a segment of sequences or a set of genes. When looking at another segment of sequences, a different phylogenetic tree can be obtained with high confidence, too. In this case, we want to measure the dissimilarity of the two trees. The rooted subtree prune and regraft (rSPR) distance of the two trees has been used for this purpose [8].

Unfortunately, it is NP-hard to compute the rSPR distance of two given phylogenetic trees [4,8]. So, it is challenging to develop programs that can compute the rSPR distance of two given trees of large rSPR distance. Indeed, this has motivated researchers to design approximation algorithms [2,3,8,11] or exact algorithms [4,13,15,14], as well as heuristic algorithms [1,7,9,10,12], for computing the rSPR distance of two given trees. The previously fastest exact algorithm is due to Whidden *et al.* [14] and runs in $O\left(2.415^d n\right)$ time, where n and d are

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 36–47, 2013.

the number of leaves and the rSPR distance of the input trees, respectively. Recently, we have shown that a fast exact algorithm for rSPR distance can be used to speed up the computation of hybridization number and the construction of minimum hybridization networks [5].

In this paper, we present a faster exact algorithm which runs in $O\left(2.344^d n\right)$ time. Our algorithm builds on Whidden *et al.*'s but requires a number of new ideas. The main idea is to repeatedly find a pair of sibling leaves in one of the input trees more carefully. We have implemented the algorithm and also applied it to other related problems such as the problem of computing the hybridization number of two or more phylogenetic trees. The experimental results are presented in [6], where a bug in the version v1.1.0 of Whidden *et al.*'s *RSPR* is also pointed out. Subsequently, Whidden *et al.* fixed the bug and released the newest version (namely, v1.1.1) of *RSPR* very recently. In the release of the buggy version v1.1.0, they also conjecture that their algorithm runs in $O\left(2^d n\right)$ time. Unfortunately, the corrected version v1.1.1 is slower than v1.1.0 and we doubt that their conjecture is true. Indeed, the running time of the previously fastest algorithm for rSPR distance remained to be $O\left(2.415^d n\right)$. Although the running time $O\left(2.344^d n\right)$ of our new algorithm may look marginally better than $O\left(2.415^d n\right)$, our experimental results show that the new algorithm is significantly faster than the newest version of Whidden *et al.*'s *RSPR*.

The remainder of this paper is organized as follows. In Section 2, we give the basic definitions that will be used throughout the paper. In Section 3, we sketch Whidden *et al.*'s algorithm for rSPR distance, because our new algorithm will build on theirs. In Section 4, we outline our algorithm. In Section 5, we detail the main step of our algorithm for an illustrative case. Due to page limit, the remainder of our algorithm is omitted and so is the analysis of its time complexity. Finally, we compare the performance of our new algorithm with that of Whidden *et al.*'s *RSPR* in Section 7.

2 Preliminaries

Throughout this paper, a *rooted forest* always means a directed acyclic graph in which every node has in-degree at most 1 and out-degree at most 2.

Let F be a rooted forest. The *roots* (respectively, *leaves*) of F are those nodes whose in-degrees (respectively, out-degrees) are 0. The *size* of F, denoted by $|F|$, is the number of roots in F minus 1. A node v of F is *unifurcate* if it has only one child in F. If a root v of F is unifurcate, then *contracting* v *in* F is the operation that modifies F by deleting v. If a non-root node v of F is unifurcate, then *contracting* v *in* F is the operation that modifies F by first adding an edge from the parent of v to the child of v and then deleting v.

For convenience, we view each node u of F as an ancestor and descendant of itself. For a node v of F, the *subtree* F_v *of* F *rooted at* v is the subgraph of F whose nodes are the descendants of v in F and whose edges are those edges connecting two descendants of v in F. If v is a non-root node of F, then F_v is called a *pendant subtree* of F. On the other hand, if v is a root of F, then F_v is a

component tree of F. F is a *rooted tree* if it has only one root. If u and v are two leaves in the same component tree of F, then the *pendant subtrees of F between u and v* are the pendant subtrees whose roots w satisfy that the (undirected) path between u and v in F does not contain w but contains the parent of w. If U is a set of leaves in a component tree of F, then $\mathrm{LCA}_F U$ denotes the lowest common ancestor (LCA) of the leaves in U.

A *rooted binary forest* is a rooted forest in which the out-degree of every non-leaf node is 2. Let F be a rooted binary forest. F is a *rooted binary tree* if it has only one root. If v is a non-root node of F with parent p and sibling u, then *detaching the pendant subtree with root v* is the operation that modifies F by first deleting the edge (p, v) and then contracting p. A *detaching operation* on F is the operation of detaching a pendant subtree of F. If v is a root leaf of F, then *eliminating v* from F is the operation that modifies F by simply deleting v. On the other hand, if v is a non-root leaf of F, then *eliminating v* from F is the operation that modifies F by first detaching the subtree rooted at v and then deleting v.

A *phylogenetic tree* on a set X of species is a rooted binary tree whose leaf set is X. Let T_1 and T_2 be two phylogenetic trees on the same set X of species. If we can apply a sequence of detaching operations on each of T_1 and T_2 so that they become the same forest F, then we refer to F as an *agreement forest* (AF) of T_1 and T_2. A *maximum agreement forest* (MAF) of T_1 and T_2 is an agreement forest of T_1 and T_2 whose size is minimized over all agreement forests of T_1 and T_2. The size of an MAF of T_1 and T_2 is called the *rSPR distance* between T_1 and T_2.

3 Sketch of Whidden et al.'s Algorithm for rSPR Distance

In this section, we sketch the previously fastest algorithm (due to Whidden et al. [14]) for computing the rSPR distance of two given phylogenetic trees. Whidden *et al.*'s algorithm indeed solves the following problem (denoted by rSPRDC, for convenience):

Input: (k, T_1, F_2), where k is a nonnegative integer, T_1 is a phylogenetic tree on a set X of species, and F_2 is a rooted forest obtained from some phylogenetic tree T_2 on X by performing zero or more detaching operations.

Output: "Yes" if performing k more detaching operations on F_2 leads to an AF of T_1 and T_2; "no" otherwise.

Obviously, to compute the rSPR distance between two given phylogenetic trees T_1 and T_2, it suffices to solve rSPRDC on input (k, T_1, T_2) for $k = 0, 1, 2, \ldots$ (in this order), until a "yes" is outputted.

Note that the input integer k to rSPRDC must be nonnegative. So, every time before we call an algorithm A for rSPRDC on an input (k, T_1, F_2), we need to check if $k \geq 0$. If $k > 0$, we proceed to call A on input (k, T_1, F_2); otherwise, we

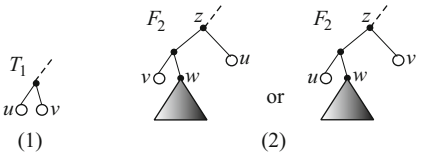


Fig. 1. The best case in Whidden *et al.*'s algorithm: (1) The subtree of T_1 rooted at the parent of u and v, (2) The subtree of F_2 rooted at the LCA z of u and v, where each black triangle indicates a pendant subtree of F_2

do not make the call. However, in order to keep the description of A simpler, we do not explicitly mention this checking process when we describe A.

Whidden *et al.*'s algorithm for rSPRDC is recursive and proceeds as follows. In the base case, $k = 0$ and it suffices to check if each component tree of F_2 is a pendant subtree of T_1. If each component tree of F_2 is a pendant subtree of T_1, then we output "yes" and stop. Otherwise, we output "no" and return. So, suppose that $k > 0$. Then, whenever T_1 has two sibling leaves u and v such that u and v are also sibling leaves in F_2, we modify T_1 and F_2 by merging u and v into a single leaf (say, u). Moreover, whenever F_2 has a root u that is also a leaf, we modify T_1 and F_2 by eliminating u from them. We repeat these two types of modifications of T_1 and F_2 until none of them is possible. After that, if F_2 becomes empty, then we can output "yes" and stop. Otherwise, we select two *arbitrary* sibling leaves u and v in T_1 and use them to distinguish three cases as follows.

Case 1: u and v are in different component trees of F_2. In this case, in order to transform T_1 and F_2 into identical forests, it suffices to try two choices to modify them, namely, by either eliminating u or eliminating v. For each choice, we further recursively solve rSPRDC on input $(k-1, T_1, F_2)$. So, we make two recursive calls here.

Case 2: u and v are in the same component tree of F_2 and either (I) u and the parent of v are siblings in F_2 or (II) v and the parent of u are siblings in F_2. See Figure 1. In this case, if (I) (respectively, (II)) holds, then in order to transform T_1 and F_2 into identical forests with the minimum number of detaching operations, it suffices to modify F_2 by detaching the subtree rooted at the sibling w of v (respectively, u), and further recursively solve rSPRDC on input $(k-1, T_1, F_2)$ [14]. So, we make only one recursive call here.

Case 3: u and v are in the same component tree of F_2 and neither (I) nor (II) in *Case 2* holds. In this case, in order to transform T_1 and F_2 into identical forests, it suffices to try three choices to modify them. The first two choices are the same as those in *Case 1*. In the third choice, we count the number b of pendant subtrees of F_2 between u and v, modify F_2 by detaching the pendant subtrees between u and v, and recursively solve rSPRDC on input $(k-b, T_1, F_2)$ if $k - b \geq 0$. Note that $b \geq 2$ and we make at most three recursive calls here.

Let $t(k)$ denote the time needed by Whidden *et al.*'s algorithm to solve rSPRDC on input (k, T_1, F_2). Whidden *et al.* [14] show that $t(k) = x^k n$, where n is the number

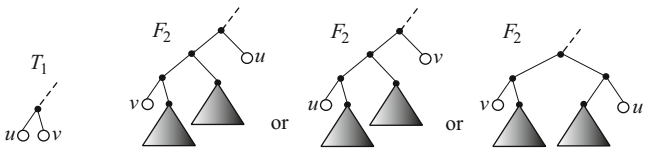


Fig. 2. The worst case in Whidden *et al.*'s algorithm, where each black triangle indicates a pendant subtree of F_2

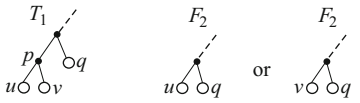


Fig. 3. The best case in our algorithm

of leaves in T_1 and x is the smallest real number satisfying the inequality $x^2 \geq 2x + 1$. Intuitively speaking, this inequality originates from *Case 3* above where $b = 2$ (see Figure 2). Since $x = 1 + \sqrt{2}$, their algorithm runs in $O\left(2.415^k n\right)$ time.

4 Ideas for Improving Whidden et al.'s Algorithm

To improve Whidden *et al.*'s algorithm, our idea is to find two sibling leaves u and v in T_1 more carefully as follows.

Step 1. *Case 2* in Section 3 is *the best case* in Whidden *et al.*'s algorithm because we make only one recursive call in this case. So, we first try to find two sibling leaves u and v (in T_1) satisfying the condition in the best case (cf. Figure 1). If such two sibling leaves u and v exist in T_1, then we use them to modify F_2 and further recursively solve rSPRDC as in Case 2 of Whidden *et al.*'s algorithm. So, we hereafter assume that such two sibling leaves u and v do not exist in T_1.

Step 2. We then try to find two sibling leaves u and v (in T_1) such that the sibling q of the parent of u and v in T_1 is also a leaf of T_1 and either (1) u and q are sibling leaves in F_2 or (2) v and q are sibling leaves in F_2 (see Figure 3). If such two sibling leaves u and v exist in T_1, then we say that *the best case* in our algorithm occurs because this case is essentially symmetric to the best case in Whidden *et al.*'s algorithm [1]. So, if such two sibling leaves u and v exist in T_1 and (1) (respectively, (2)) holds, then we can modify T_1 and F_2 by eliminating v (respectively, u) from them, and further recursively solve rSPRDC on input $(k-1, T_1, F_2)$. So, we hereafter assume that the best case in our algorithm does not occur.

Step 3. We try to find two sibling leaves u and v (in T_1) satisfying the condition in Case 1 of Whidden *et al.*'s algorithm. If such two sibling leaves u and v exist in T_1, then we use them to modify T_1 and F_2 and further recursively solve rSPRDC

[1] In other words, we can use a similar argument of [14] to prove the correctness of the processing of T_1 and F_2 in this case.

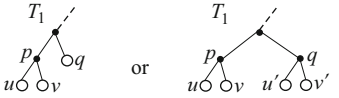


Fig. 4. The two possible cases for two sibling leaves u and v farthest from the root

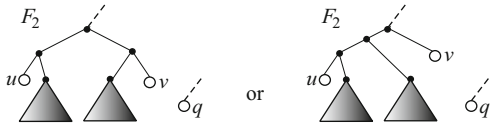


Fig. 5. The subtrees of F_2 rooted at $\mathrm{LCA}_{F_2}\{u, v\}$ and q in Case 1.1, where each black triangle indicates a pendant subtree

as in Case 1 of Whidden *et al.*'s algorithm. So, we hereafter assume that such two sibling leaves u and v do not exist in T_1.

Step 4. We try to find two sibling leaves u and v (in T_1) such that there are at least three pendant subtrees of F_2 between u and v. If such two sibling leaves u and v exist in T_1, then we use them to modify F_2 and further recursively solve rSPRDC as in Case 3 of Whidden *et al.*'s algorithm. So, we hereafter assume that such two sibling leaves u and v do not exist in T_1.

Step 5. We select two sibling leaves u and v (in T_1) whose distance from the root of T_1 is maximized. Let p be the parent of u and v in T_1, and q be the sibling of p in T_1. By our choice of u and v, it follows that either q is a leaf or both children u' and v' of q in T_1 are leaves (see Figure 4). So, we have two cases to consider, depending on whether q is a leaf or not. Section 5 details the case where q is a leaf, while Section 6 details the worst case in our algorithm. The other cases are omitted due to page limit.

5 Details of Step 5 in Section 4 When q Is a Leaf

Throughout this section, we assume that q is a leaf. Since u and v are sibling leaves in T_1, the assumptions in Steps 1 through 4 in Section 4 imply that (1) there are exactly two pendant subtrees of F_2 between u and v and (2) u and q are not sibling leaves in F_2 and neither are v and q in F_2. Since u (respectively, v) and q are not sibling leaves in F_2, the (undirected) path between u (respectively, v) and q contains at least three edges if q belongs to the component tree of F_2 that contains u and v. We can also assume that the distance from z to u is not shorter than the distance from z to v, where $z = \mathrm{LCA}_{F_2}\{u, v\}$. So, one of the following cases occurs.

Case 1.1: q does not belong to the same component tree of F_2 as u and v (see Figure 5). In this case, our idea for speeding up Whidden *et al.*'s algorithm is as follows. Basically, we emulate Whidden *et al.*'s algorithm by using the sibling

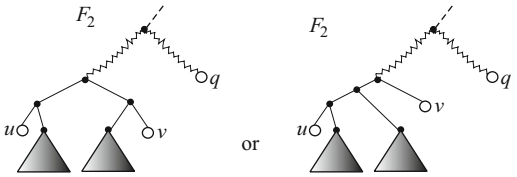


Fig. 6. A portion of F_2 in Case 1.2, where each zig-zag line indicates a path containing at least one edge and each black triangle indicates a pendant subtree

pair (u, v) to try three different choices of modifying T_1 and F_2. The first choice is to eliminate u from T_1 and F_2. A crucial point is that after the elimination of u, we know that v and q become a new sibling pair of T_1 and they belong to different component trees of F_2, implying that we need to use the pair (v, q) to further try two different choices of modifying T_1 and F_2 (namely, eliminating either v or q from T_1 and F_2). So, the first choice leads to two ways of modifying T_1 and F_2 one of which is to eliminate u and v from T_1 and F_2 and the other is to eliminate u and q from T_1 and F_2. Similarly, the second choice is to eliminate v from T_1 and F_2 and it leads to two ways of modifying T_1 and F_2 one of which is to eliminate v and u from T_1 and F_2 and the other is to eliminate v and q from T_1 and F_2. Note that the first and the second choices lead to four ways of modifying T_1 and F_2 and hence lead to four recursive calls. The really crucial point here is that the four recursive calls can be reduced to three, because two of them are identical. Finally, the third choice is to modify F_2 by detaching the pendant subtrees between u and v. In summary, in order to transform T_1 and F_2 into identical forests, it suffices to try the following four different choices to modify them and further make recursive calls:

1. We eliminate u and v from T_1 and F_2, and then recursively solve rSPRDC on input $(k-2, T_1, F_2)$.
2. We eliminate u and q from T_1 and F_2, and then recursively solve rSPRDC on input $(k-2, T_1, F_2)$.
3. We eliminate v and q from T_1 and F_2, and then recursively solve rSPRDC on input $(k-2, T_1, F_2)$.
4. We detach the two pendant subtrees of F_2 between u and v, and then recursively solve rSPRDC on input $(k-2, T_1, F_2)$.

Case 1.2: u, v, and q belong to the same component tree of F_2 and $\mathrm{LCA}_{F_2}\{u, v\}$ is not an ancestor of q in F_2 (see Figure 6). In this case, in order to transform T_1 and F_2 into identical forests, it suffices to try four different choices to modify them and further make recursive calls:

1. We eliminate u from T_1 and F_2, and then recursively solve rSPRDC on input $(k-1, T_1, F_2)$.
2. We eliminate v and q from T_1 and F_2, and then recursively solve rSPRDC on input $(k-2, T_1, F_2)$.
3. We first eliminate v from T_1 and F_2. Let b be the number of pendant subtrees of F_2 between u and q. Note that $b \geq 2$. We then detach the b pendant

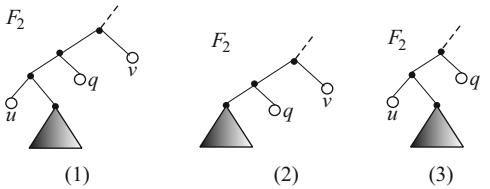


Fig. 7. (1) The subtree of F_2 rooted at $\mathrm{LCA}_{F_2}\{u, v\}$ in Case 1.3, (2) the situation in Case 1.3 immediately after eliminating u from F_2, and (3) the situation in Case 1.3 immediately after eliminating v from F_2, where each black triangle indicates a pendant subtree

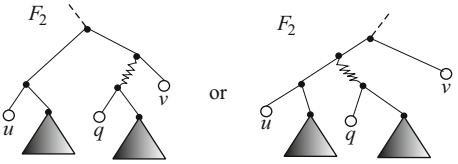


Fig. 8. A portion of F_2 in Case 1.4, where each zig-zag line indicates a path containing at least one edge and each black triangle indicates a pendant subtree

subtrees of F_2 between u and q. After that, we recursively solve rSPRDC on input $(k - 1 - b, T_1, F_2)$.
4. We detach the two pendant subtrees of F_2 between u and v, and then recursively solve rSPRDC on input $(k - 2, T_1, F_2)$.

Case 1.3: u, v, and q belong to the same component tree of F_2, $\mathrm{LCA}_{F_2}\{u, v\}$ is an ancestor of q in F_2, the (undirected) path between u and q in F_2 contains exactly three edges and so does the (undirected) path between v and q in F_2 (see Figure 7(1)). In order to transform T_1 and F_2 into identical forests, it suffices to try three difference choices to modify them and further make recursive calls:

1. We first eliminate u from T_1 and F_2 (see Figure 7(2)). Then, the best case in Whidden *et al.*'s algorithm occurs. More precisely, we can further detach the subtree of F_2 rooted at the sibling of q. After that, we recursively solve rSPRDC on input $(k - 2, T_1, F_2)$.
2. We first eliminate v from T_1 and F_2 (see Figure 7(3)). Then, the best case in Whidden *et al.*'s algorithm occurs. More precisely, we can further detach the subtree of F_2 rooted at the sibling of u. After that, we recursively solve rSPRDC on input $(k - 2, T_1, F_2)$.
3. We detach the two pendant subtrees of F_2 between u and v, and further recursively solve rSPRDC on input $(k - 2, T_1, F_2)$.

Case 1.4: u, v, and q belong to the same component tree of F_2, $\mathrm{LCA}_{F_2}\{u, v\}$ is an ancestor of q in F_2, and the (undirected) path between u and q in F_2 contains at least four edges and is not shorter than the (undirected) path between v and q in F_2 (see Figure 8). In this case, we proceed as in Case 1.2.

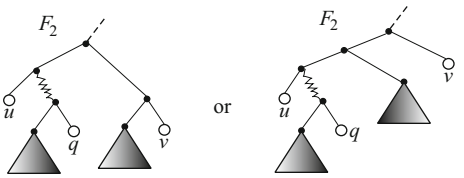


Fig. 9. A portion of F_2 in Case 1.5, where each zig-zag line indicates a path containing at least one edge and each black triangle indicates a pendant subtree

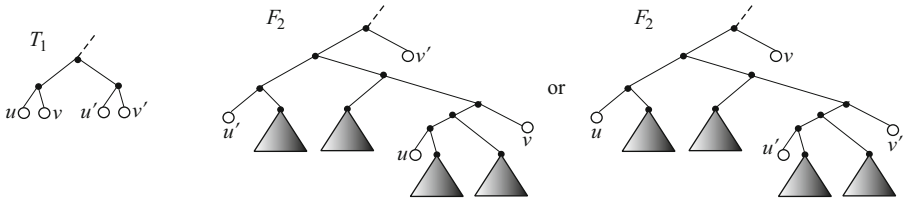


Fig. 10. The worst configuration of u, v, u', and v' in our algorithm

Case 1.5: u, v, and q belong to the same component tree of F_2, $\mathrm{LCA}_{F_2}\{u, v\}$ is an ancestor of q in F_2, and the (undirected) path between u and q in F_2 is shorter than the (undirected) path between v and q in F_2 (see Figure 9). In this case, in order to transform T_1 and F_2 into identical forests, it suffices to try four different choices to modify them and further make recursive calls:

1. We eliminate v from T_1 and F_2, and then recursively solve rSPRDC on input $(k-1, T_1, F_2)$.
2. We eliminate u and q from T_1 and F_2, and then recursively solve rSPRDC on input $(k-2, T_1, F_2)$.
3. We first eliminate u from T_1 and F_2. Let b be the number of pendant subtrees of F_2 between v and q. Note that $b \geq 2$. We then detach the b pendant subtrees of F_2 between v and q. After that, we recursively solve rSPRDC on input $(k-1-b, T_1, F_2)$.
4. We detach the two pendant subtrees of F_2 between u and v, and then recursively solving rSPRDC on input $(k-2, T_1, F_2)$.

6 The Worst Case in Our Algorithm

The worst case in our algorithm occurs when u, v, u' and v' are located in T_1 and F_2 as shown in Figure 10.

In this case, in order to transform T_1 and F_2 into identical forests, it suffices to try fifteen different choices to modify them and make recursive calls as follows.

1. We modify T_1 and F_2 as follows. First, we eliminate u and u' from T_1 and F_2. Then, v and v' become two sibling leaves in T_1. Let b be the number of pendant subtrees of F_2 between v and v'. The crucial point is that $b = 3$.

We proceed to further modify T_1 and F_2 in three different ways and accordingly make three recursive calls as in Case 3 in Whidden *et al.*'s algorithm. In more detail, the three recursive calls solve rSPRDC on input $(k-3, T_1, F_2)$, $(k-3, T_1, F_2)$, and $(k-5, T_1, F_2)$, respectively. Intuitively speaking, we are able to avoid the worst case in Whidden *et al.*'s algorithm when processing the sibling leaves v and v'.

2. We modify T_1 and F_2 similarly as in Item 1. The only difference is that we first eliminate u and v' from T_1 and F_2 and then proceed to use the sibling leaves v and u' to further modify T_1 and F_2 in three different ways and accordingly make three recursive calls as in Case 3 in Whidden *et al.*'s algorithm.
3. We modify T_1 and F_2 as follows. First, we eliminate u from T_1 and F_2. Then, we modify F_2 by detaching the pendant subtrees between u' and v'. After these modifications, we recursively solve rSPRDC on input $(k-3, T_1, F_2)$.
4. We modify T_1 and F_2 as follows. First, we eliminate v and u' from T_1 and F_2. Then, u and v' become two sibling leaves in T_1. Let b be the number of pendant subtrees of F_2 between u and v'. The crucial point is that $b = 4$. We proceed to further modify T_1 and F_2 in three different ways and accordingly make three recursive calls as in Case 3 in Whidden *et al.*'s algorithm. In more detail, the three recursive calls solve rSPRDC on input $(k-3, T_1, F_2)$, $(k-3, T_1, F_2)$, and $(k-6, T_1, F_2)$, respectively. Intuitively speaking, we are able to avoid the worst case in Whidden *et al.*'s algorithm when processing the sibling leaves u and v'.
5. We modify T_1 and F_2 similarly as in Item 4. The only difference is that we first eliminate v and v' from T_1 and F_2 and then proceed to use the sibling leaves u and u' to further modify T_1 and F_2 in three different ways and accordingly make three recursive calls as in Case 3 in Whidden *et al.*'s algorithm.
6. We modify T_1 and F_2 as follows. First, we eliminate v from T_1 and F_2. Then, we modify F_2 by detaching the pendant subtrees between u' and v'. After these modifications, we recursively solve rSPRDC on input $(k-3, T_1, F_2)$.
7. We modify F_2 by detaching the pendant subtrees between u and v. After these modifications, we recursively solve rSPRDC on input $(k-2, T_1, F_2)$.

The execution of our algorithm on input (k, T_1, F_2) can be modeled by a tree Γ in which the root corresponds to (k, T_1, F_2), each other node corresponds to a recursive call, and a recursive call A is a child of another call B if and only if B calls A directly. We call Γ the *search tree* on input (k, T_1, F_2). For convenience, we define the *size* of Γ to be the number of its nodes.

Let $s(k)$ denote the maximum size of a search tree on input (k, T_1, F_2), where the maximum is taken over all T_1 and F_2. Clearly, $s(0) = 1$. Let k be an arbitrary positive integer. In the above worst case, we have $s(k) \leq s(k-2) + 10s(k-3) + 2s(k-5) + 2s(k-6) + 1$. By induction on k, we can show that $s(k) \leq 2.344^{k+1} - 1$.

Theorem 1. *Given two phylogenetic trees T_1 and T_2 with the same leaf set, we can compute the rSPR distance of T_1 and T_2 in $O\left(2.344^k n\right)$ time, where n is the number of leaves in the input trees.*

7 Performance Comparison

We have implemented our algorithm to obtain a program for computing the rSPR distance of two given phylogenetic trees. To compare the efficiency of our program with that of the previous best (namely, Whidden *et al.*'s program), we have run them on simulated datasets. The experiments have been performed on a Windows-7 (x64) laptop PC with i7-M640 2.8GHz CPU and 4GB RAM.

To generate simulated datasets, we use a program due to Beiko and Hamilton [1]. To obtain a pair (T, T') of trees each with n (say, 100) leaves, their program first generates T randomly and then obtains T' from T by performing a specified number r (say, 50) of random rSPR operations on T. Recall that an *rSPR operation* on a tree T first removes an edge (p, c) from T, then contracts p (the vertex of out-degree 1 resulting from the removal of edge (p, c)), and further re-attaches the subtree rooted at c to an edge (p', c') of T (by introducing a new vertex m', splitting edge (p', c') into two edges (p', m') and (m', c'), and adding a new edge (m', c)). So, the actual rSPR distance of T and T' is at most r.

We use Beiko and Hamilton's program to generate 120 pairs of trees by setting $n = 100$ and $r = 50$[2]. It turns out that among the 120 generated tree-pairs, 4 (respectively, 10, 23, 17, 30, 17, 9, 5, 2, or 2) tree-pairs have rSPR distance 39

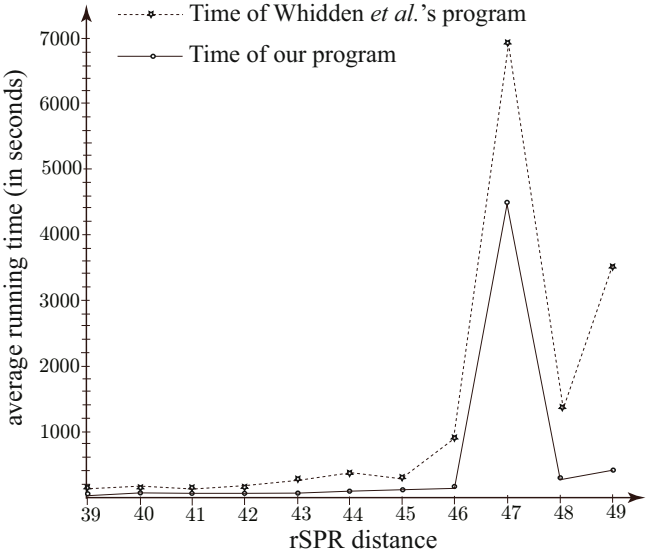


Fig. 11. Comparing our program against Whidden *et al.*'s on 120 randomly generated tree-pairs, where each tree has **100** leaves

[2] For smaller r (say, $r = 40$), there is no much difference in speed between our algorithm and Whidden et al.'s.

(respectively, 40, 41, 42, 43, 44, 45, 46, 47, or 48). Moreover, there is exactly one tree-pair with rSPR distance 49. Figure 11 summarizes the average running times of our program and Whidden *et al.*'s for computing the rSPR distances of the generated tree-pairs, where each average is taken over those tree-pairs with the same rSPR distance. As can be seen from the figure, our program is significantly faster than Whidden *et al.*'s. This difference in speed becomes more significant as the rSPR distance becomes larger.

References

1. Beiko, R.G., Hamilton, N.: Phylogenetic identification of lateral genetic transfer events. BMC Evol. Biol. 6, 159–169 (2006)
2. Bonet, M.L., St. John, K., Mahindru, R., Amenta, N.: Approximating subtree distances between phylogenies. Journal of Computational Biology 13, 1419–1434 (2006)
3. Bordewich, M., McCartin, C., Semple, C.: A 3-approximation algorithm for the subtree distance between phylogenies. Journal of Discrete Algorithms 6, 458–471 (2008)
4. Bordewich, M., Semple, C.: On the computational complexity of the rooted subtree prune and regraft distance. Annals of Combinatorics 8, 409–423 (2005)
5. Chen, Z.-Z., Wang, L.: FastHN: A Fast Tool for Minimum Hybridization Networks. BMC Bioinformatics 13, 155 (2012)
6. Chen, Z.-Z., Wang, L.: An Ultrafast Tool for Minimum Reticulate Networks. Journal of Computational Biology 20(1), 38–41 (2013)
7. Goloboff, P.A.: Calculating SPR distances between trees. Cladistics 24(4), 591–597 (2007)
8. Hein, J., Jing, T., Wang, L., Zhang, K.: On the complexity of comparing evolutionary trees. Discrete Appl. Math. 71, 153–169 (1996)
9. Hill, T., Nordström, K.J., Thollesson, M., Säfström, T.M., Vernersson, A.K., Fredriksson, R., Schiöth, H.B.: SPRIT: Identifying horizontal gene transfer in rooted phylogenetic trees. BMC Evolutionary Biology 10, 42 (2010)
10. MacLeod, D., Charlebois, R.L., Doolittle, F., Bapteste, E.: Deduction of probable events of lateral gene transfer through comparison of phylogenetic trees by recursive consolidation and rearrangement. BMC Evolutionary Biology 5(1), 27 (2005)
11. Rodrigues, E.M., Sagot, M.-F., Wakabayashi, Y.: The maximum agreement forest problem: Approximation algorithms and computational experiments. Theoretical Computer Science 374, 91–110 (2007)
12. Than, C., Ruths, D., Nakhleh, L.: PhyloNet: a software package for analyzing and reconstructing reticulate evolutionary relationships. BMC Bioinformatics 9, 322 (2008)
13. Wu, Y.: A practical method for exact computation of subtree prune and regraft distance. Bioinformatics 25(2), 190–196 (2009)
14. Whidden, C., Beiko, R.G., Zeh, N.: Fast FPT algorithms for computing rooted agreement forests: Theory and experiments. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 141–153. Springer, Heidelberg (2010)
15. Whidden, C., Zeh, N.: A unifying view on approximation and FPT of agreement forests. In: Salzberg, S.L., Warnow, T. (eds.) WABI 2009. LNCS, vol. 5724, pp. 390–402. Springer, Heidelberg (2009)

Arbitrated Quantum Signature Schemes: Attacks and Security

Xiangfu Zou[1,2] and Daowen Qiu[2,3,⋆]

[1] School of Mathematics and Computational Science,
Wuyi University, Jiangmen 529020, China
[2] Department of Computer Science,
Sun Yat-sen University, Guangzhou 510006, China
[3] SQIG–Instituto de Telecomunicações, Departamento de Matemática,
Instituto Superior Técnico, Technical University of Lisbon,
Av. Rovisco Pais 1049-001, Lisbon, Portugal
issqdw@mail.sysu.edu.cn

Abstract. In this paper, we first summarize the attacks on the existing *arbitrated quantum signature* (AQS) schemes and then present a valid forgery attack. Also, we discuss the effectiveness of these attacks and analyze the reasons for these schemes suffered attacks. Moreover, we propose an AQS scheme which can resist all existent attacks. The proposed AQS scheme can preserve all merits in the previous AQS schemes such as it can sign the known and unknown quantum messages. To achieve higher security of AQS, we also construct a strong quantum one-time pads encryption which is applied to improve the AQS schemes.

Keywords: Quantum cryptography, digital signature, arbitrated quantum signature, security, attacks.

1 Introduction

Digital signatures can be divided into two categories: direct digital signatures and arbitrated digital signatures. With the help of an arbitrator, an arbitrated digital signature scheme is able to complete some impossible tasks of a direct digital signature scheme. For example, Barnum *et al.* [2] pointed out that digitally signing quantum states by non-interactive protocol is impossible. However, Zeng and Keitel [30] suggested an *arbitrated quantum signature* (AQS) scheme which can sign the known and unknown quantum messages. Because this AQS scheme can sign quantum messages, it was further discussed [12, 23, 29, 31]. Note that, the authors [31] pointed out that the existing AQS schemes [23, 30] can be repudiated by the receiver Bob, and improved AQS schemes to conquer this shortcoming.

Very recently, some attacks [11, 17, 28] on the AQS schemes [23, 30, 31] was proposed. Gao *et al.* [17] showed that in the AQS protocols [23, 31] the receiver

⋆ Corresponding author.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 48–59, 2013.

Bob can realize existential forgery of the sender's signature under known message attacks and the sender Alice can successfully disavow any of her signatures by a simple attack. Choi *et al.* [11] pointed out that there exists a simple existential forgery attack on the AQS scheme [30] to validly modify the transmitted pair of message and signature. Sun *et al.* [28] showed that in all the presented AQS protocols [23, 30, 31] Alice can always successfully acquire the receiver Bob's secret key and disavow any of her signatures.

Because there are so many attacks [11, 17, 28] on AQS protocols, some improvements on these AQS protocols are urgently needed. Though these literatures [11, 17, 28] discussed about the potential improvements of AQS and suggested some improved AQS protocols, the improvements can not resist all existent attacks [11, 17, 28]. In this paper, we will first analyze the effectiveness of these attacks and the reasons why the AQS protocols are susceptible to attacks. Then, we will propose an AQS scheme which is secure against all existent attacks [11, 17, 28].

2 Quantum One-Time Pads Encryption

To discuss and improve the security of AQS, we recall the quantum one-time pads encryption and construct a strong quantum one-time pads encryption which is applied to improve the AQS schemes.

The *quantum one-time pads* (QOTP) encryption was proposed by Boykin and Roychowdhury [7]. For convenience, the symbol E_K denotes the QOTP encryption according to some key $K \in \{0,1\}^*$ satisfying $|K| \geq 2n$ as follows:

$$E_K(|P\rangle) = \bigotimes_{i=1}^{n} \sigma_x^{K_{2i-1}} \sigma_z^{K_{2i}} |P_i\rangle, \tag{1}$$

where K_j denotes the j-th bit of K, $|P\rangle$ a quantum message as $|P\rangle = |P_1\rangle \otimes |P_2\rangle \otimes \cdots \otimes |P_n\rangle$ with $|P_i\rangle = \alpha_i|0\rangle + \beta_i|1\rangle$, $\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and $\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ Pauli matrices.

In this paper, we use $\mathbb{E}_K$ to denote a new QOTP encryption according to some key $K \in \{0,1\}^*$ satisfying $|K| \geq 3n$ as follows:

$$\mathbb{E}_K(|P\rangle) = \bigotimes_{i=1}^{n} \theta^{K_{3i-2}} \sigma_x^{K_{3i-1}} \sigma_z^{K_{3i}} |P_i\rangle, \tag{2}$$

where $\theta = \frac{1}{2} \begin{pmatrix} \sqrt{2} & \sqrt{2} \\ 1+\mathbb{i} & -1-\mathbb{i} \end{pmatrix}$, $\mathbb{i}$ and $-\mathbb{i}$ the square roots of -1. For convenience, we call the new QOTP *strong quantum one-time pads* (SQOTP).

By direct verifying, the SQOTP encryption satisfies the necessary and sufficient condition for security, Lemma 5.1 in Ref. [7], i.e.,

$$\frac{1}{8} \sum_{K=(000)_2}^{(111)_2} U_K \sigma_x^{\alpha} \sigma_z^{\beta} U_K^{\dagger} = \delta_{\alpha,0} \delta_{\beta,0} I, \tag{3}$$

where $\alpha, \beta \in \{0,1\}$, I the second-order identity matrix, $U_K = \theta^{K_1}\sigma_x^{K_2}\sigma_z^{K_3}$, $K \in \{0,1\}^3$ and $(\cdots)_2$ binary number.

Let U be a second-order matrix. We can obtain the following property of SQOTP encryption by simple computation.

Proposition 1. If the following equations

$$\mathbb{E}_K(U|P\rangle) = \lambda_K U\mathbb{E}_K(|P\rangle),\ K \in \{0,1\}^3 \tag{4}$$

hold for any quantum state $|P\rangle = \alpha|0\rangle + \beta|1\rangle$, then U must be $c\begin{pmatrix}1&0\\0&1\end{pmatrix}$ where c is any complex number, and $\lambda_K = 1$ for any $K \in \{0,1\}^3$.

Note that, the QOTP encryption [7, 23, 31] does not satisfy Proposition 1. To satisfy Proposition 1, the SQOTP encryption needs $3n$ bits key to encrypt an n-bit message.

3 Attacks on Arbitrated Quantum Signature

In this section, we will recall and reappraise the attacks on AQS schemes.

3.1 Bob's Forgery Attacks

A secure arbitrated quantum signature scheme should satisfy that the signature should not be forged by the attacker (including the malicious receiver) [23, 30, 31]. Gao *et al.* [17] showed that in the AQS protocols [23, 31] the receiver Bob can realize existential forgery. By Gao's attacks [17], Bob can forge Alice's signature by tampering a existing Alice's signature according to a new message after he received a valid message-signature pair. Similarly, Choi *et al.* [11] pointed out that there exists an existential forgery attack on the AQS scheme [30]. The both attacks [11, 17] are that Bob chooses an appropriate unitary matrix $U = \bigotimes_{i=i}^{n} U_i$ such that

$$U_i E_{K_{2i-1}K_{2i}}(|P_i\rangle) = cE_{K_{2i-1}K_{2i}}(U_i|P_i\rangle) \tag{5}$$

for $i = 1, 2, \ldots, n$, where c is a complex number with $|c| = 1$. Then, he can use US_A as the Alice's signature for the new message $U|P\rangle$.

Indeed, there exists another forgery attack in our former AQS schemes [31] for the receiver Bob. Bob chooses any $2n$ number r' and gets

$$|P_B\rangle = E_{r'}^{-1}(E_r(|P\rangle)). \tag{6}$$

Then, Bob can use $(|S_A\rangle, r')$ as the signature of the message $|P_B\rangle$.

3.2 Alice's Disavowal Attacks

A secure arbitrated quantum signature scheme should satisfy the impossibility of disavowal by the signatory [23, 30, 31]. Gao *et al.* [17] showed that, in the AQS protocols [23, 31], the sender Alice can successfully disavow her signature by tampering the signature when Trent sends it back to Bob. It is easy to know

that this disavowal attack can be used successfully to the AQS scheme using GHZ states [30]. In addition, Sun *et al.* [28] show that in all the presented AQS protocols [23, 30, 31] Alice can always successfully acquire the receiver Bob's secret key and disavow any of her signatures. Hwang *et al.* [19] proposed an *invisible photon eavesdropping* (IPE) Trojan horse attack such that Alice can acquire the receiver Bob's secret key and disavow her signatures.

3.3 The Reasons for AQS Scheme Suffered Attacks

From the discussion above, all existing AQS schemes [23, 30, 31] are susceptible to known message attacks and Alice's disavowal attacks. By the principles of the attacks and the discussion in Ref. [17], the following three facts are the main reasons why the AQS protocols are susceptible to the attacks.

(1) The arbitrator Trent does not know the content of the signed message because it is encoded with quantum states. This fact gives the chance for Bob to tamper the message and its signature synchronously without being discovered.

(2) Pauli operations are commute or anticommute with each other. This fact makes that the message and the signature still can pass Trent's verification after the attacker using same Pauli operations on them. Therefore, Bob can give many existential forgeries based on one legal signed message.

(3) When Trent sends Alice's signature back to Bob, it is totally unreadable for Bob and its integrity cannot be verified. This gives Alice a chance to intercept and modify her signature without being discovered, and then successfully disavow her signature.

3.4 Other Attacks

We [31] found that the integrity of the signature in Refs. [23, 30] can be repudiated by the receiver Bob. For convenience, we call it *deny-integrity attack*. To conquer the deny-integrity attack, we [31] proposed two improved AQS schemes.

Hwang *et al.* [19] thought that in our previous AQS schemes [31] Bob can also repudiate the integrity of the signature by a special case of DoS attack, and the signer can reveal the verifier's secret key by Trojan horse attacks.

First, we point out that the "deny-integrity attack" proposed by Hwang *et al.* [19] is not a real deny-integrity attack. The deny-integrity attack must be performed after Bob receives integrated Alice's message-signature pair, while Hwang's "deny-integrity attack" is performed before Bob receives integrated Alice's message-signature pair.

Secondly, we clarify that the "DoS attack" proposed by Hwang *et al.* [19] is not a real DoS attack. The DoS attack is characterized by an explicit attempt by some attackers to prevent legitimate users from using resources [22]. However, it is not a DoS attack that legitimate users do not perform their duties. Therefore, Bob initiatively gives up receiving Alice signed message in Ref. [19] is not a real deny-integrity attack.

Finally, we consider the Trojan horse attack. The first Trojan horse attack on quantum cryptography was introduced by Gisin *et al.* [18]. Also, they pointed out that there is no doubt that Trojan horse attacks can be prevented by technical measures. The second Trojan horse attack is the IPE attack proposed by Cai [10]. Note that, Cai [10] also proposed a method against the second Trojan horse attack, i.e., Alice adds a wavelength filter on each signal before she deals with it (i.e., codes or measures it). The third one is the delay-photon Trojan horse attack presented by Deng *et al.* [14]. To resist delay-photon Trojan horse attack, Alice should use a *photon number splitter* (PNS) to divide each sample signal into two pieces and measure them with two measuring bases. The Trojan horse attacks mentioned in Ref. [19] are not new ones. They are just the IPE attack and the delay-photon attack. To resist the Trojan horse attacks, the AQS schemes need not do any theoretic modification. We only need to pay attention to their practical implementation, i.e., the entries need to use a wavelength filter and a PNS before they deal with the photons when the AQS schemes are realized by photons.

4 An AQS Scheme

Because there are so many attacks [11, 17, 28] on AQS protocols, some improvements on these AQS protocols are urgently needed. Though these literatures [11, 17, 28] discussed about the potential improvements of AQS and suggested some improved AQS protocols, the improvements can not resist all existent attacks [11, 17, 28]. Based on these suggestions [11, 17, 28] and the previous discussion about attacks, we proposed an AQS scheme which is secure against all existent attacks [11, 17, 28].

Note that QKD schemes [3, 4, 16] and semiquantum key distribution protocols [5, 6, 32] utilize a public board or a classical public communication channel that are assumed to be susceptible to eavesdropping but not to be the injection or alteration of messages [3–6, 16, 32]. Also, we use a public board (or a classical public communication channel) that can not be blocked to recall the security of AQS schemes.

The proposed AQS scheme also involve three participants: the signatory Alice, the receiver Bob, and the arbitrator Trent, and three phases: the initializing phase, the signing phase, and the verifying phase. Suppose Alice need to sign the quantum message $|P\rangle = |P_1\rangle \otimes |P_2\rangle \otimes \cdots \otimes |P_n\rangle$ with $|P_i\rangle = \alpha_i|0\rangle + \beta_i|1\rangle$ and has at least three copies of $|P\rangle$. For obtaining a low enough error probability in the verifying phase, we can use $|P\rangle^{\otimes m}$ instead of $|P\rangle$ where m is a large enough integer. The AQS scheme using Bell states is specified in the following.

Initializing Phase

Step I1. Alice shares a $3n$ bits secret key K_{AT} with the arbitrator Trent by the quantum key distribution protocols [3, 4, 16] that were proved to be unconditionally secure [18, 20, 24, 25, 27]. Similarly, Bob shares a $3n$ bits secret key K_{BT} with Trent.

Step I2. Alice generates n Bell states $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle$ with $|\psi_i\rangle = \frac{1}{\sqrt{2}}(|00\rangle_{AB} + |11\rangle_{AB})$, where the subscripts A and B correspond to Alice and Bob, respectively. Then, Alice distributes one particle of each Bell state to Bob by employing a secure and authenticated method [2, 13]. Using quantum repeaters [8, 15] and fault-tolerant quantum computation [1, 26], Alice and Bob can share n Bell states of almost perfect fidelity even if they are far away from each other [24].

Signing Phase

Step S1. Alice randomly chooses a number $R \in \{0,1\}^{3n}$ and transforms $|P\rangle$ into an secret qubit string $|P'\rangle = \mathbb{E}_R(|P\rangle)$ by the SQOTP encryption as follows:

$$\mathbb{E}_R(|P\rangle) = \bigotimes_{i=1}^{n} \theta^{R_{3i-2}} \sigma_x^{R_{3i-1}} \sigma_z^{R_{3i}} |P_i\rangle, \tag{7}$$

where $\theta = \frac{1}{2}\begin{pmatrix} \mathbb{i}\sqrt{2} & \sqrt{2} \\ 1+\mathbb{i} & -1-\mathbb{i} \end{pmatrix}$, $\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$, $\mathbb{i}$ and $-\mathbb{i}$ the square roots of -1.

Step S2. Similarly, Alice generates $|S_A\rangle = \mathbb{E}_{K_{AT}}(|P'\rangle)$ by the SQOTP encryption.

Step S3. Alice combines each secret message state $|P'_i\rangle$ and the Bell state $|\psi_i\rangle$, and obtains the three-particle entangled state,

$$|\phi_i\rangle = |P'_i\rangle \otimes |\psi_i\rangle \tag{8}$$

$$\begin{aligned} &= \frac{1}{2}[|\Psi_{12}^+\rangle_A(\alpha'_i|0\rangle_B + \beta'_i|1\rangle_B) + |\Psi_{12}^-\rangle_A(\alpha'_i|0\rangle_B - \beta'_i|1\rangle_B) \\ &\quad + |\Phi_{12}^+\rangle_A(\beta'_i|0\rangle_B + \alpha'_i|1\rangle_B) + |\Phi_{12}^-\rangle_A(\beta'_i|0\rangle_B - \alpha'_i|1\rangle_B)], \end{aligned} \tag{9}$$

where $|\Psi_{12}^+\rangle_A, |\Psi_{12}^-\rangle_A, |\Phi_{12}^+\rangle_A$, and $|\Phi_{12}^+\rangle_A$ are the four Bell states [21], and $|P'\rangle = \alpha'_i|0\rangle_B + \beta'_i|1\rangle_B$.

Step S4. Alice implements Bell measurement on her two particles of each three-particle entangled state $|\phi_i\rangle$ and obtains $|\mathcal{M}_A\rangle = (|\mathcal{M}_A^1\rangle, |\mathcal{M}_A^2\rangle, \ldots, |\mathcal{M}_A^n\rangle)$, where $|\mathcal{M}_A^i\rangle$ represents one of the four Bell states.

Step S5. Alice sends $|S\rangle = (|P'\rangle, |S_A\rangle, |\mathcal{M}_A\rangle)$ to Bob.

Verifying Phase

Step V1. Bob sends $|Y_B\rangle = (|P'\rangle, |S_A\rangle)$ to Trent after he receives $|S\rangle$.

Step V2. Trent encrypts $|P'\rangle$ with K_{AT} by the SQOTP encryption and obtains $|S_T\rangle$ which should be consistent with $|S_A\rangle$. If $|S_T\rangle = |S_A\rangle$, the arbitrator continues the next step; Otherwise, he informs Alice and Bob by public board that the scheme aborts.

This step includes quantum state comparison. The technique of comparing two unknown quantum states was first presented in Ref. [9] and discussed in Refs. [23, 31].

Step V3. Trent gets back $|P'\rangle$ from one of $|S_A\rangle$ (i.e., $|S_T\rangle$). Then, he encrypts $|P'\rangle$ with K_{BT} and obtains $|S_B\rangle = \mathbb{E}_{K_{BT}}(|P'\rangle)$. Subsequently, he gets $|Y_T\rangle$ by reordering randomly the qubits in $|S_A\rangle$ and $|S_B\rangle$, and sends $|Y_T\rangle$ to Bob. Trent does not publish the encoding order of $|Y_T\rangle$ by the public board until Bob receives $|Y_T\rangle$.

Step V4. Bob obtains $|S_A\rangle$ and $|S_B\rangle$ by reordering $|Y_T\rangle$, and gets $|P'\rangle$ by decrypting $|S_B\rangle$ with the key K_{BT}.

Step V5. According to Alice's measurement outcomes $|\mathcal{M}_A\rangle$ and the principle of teleportation, Bob obtains $|P'_B\rangle$. Then he makes comparisons between $|P'_B\rangle$ and $|P'\rangle$. Here the way of comparing $|P'_B\rangle$ and $|P'\rangle$ is the same as that of comparing $|S_T\rangle$ and $|S_A\rangle$ in Step V2. If $|P'_B\rangle \neq |P'\rangle$, Bob rejects the signature; Otherwise, he informs Alice by the public board to publish the random number R.

Step V6. Alice publishes R by the public board.

Step V7. Bob gets back $|P\rangle = \mathbb{E}_R^{-1}(|P'\rangle)$ from $|P'\rangle$ with R and holds $(|S_A\rangle, R)$ as Alice's signature for the quantum message $|P\rangle$. The arbitrator Trent records the signature key K_{AT} and the random number R to prevent dissensions.

If the improved AQS scheme states ends normally, its communications are described in Fig. 1.

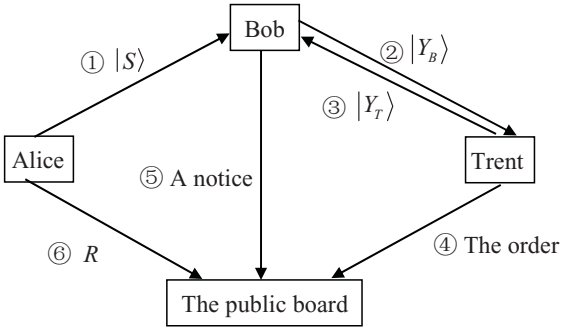


Fig. 1. The communications of the AQS scheme using Bell states

5 Security Analyses

A secure quantum signature scheme should satisfy two requirements [23, 29–31]: (1) The signature should not be forged by the attacker (including the malicious receiver); (2) The signature should not be disavowed by the signatory and the receiver. In this section, we will show that the proposed AQS scheme satisfy the two requirements above. In particular, we will show that it is secure against all existing attacks [11, 17, 28].

5.1 The Method to Resolve Disputes

To better understand the proposed AQS scheme, we clarify the ways for the arbitrator Trent to resolve disputes between the signer Alice and the receiver Bob. To resolve disputes, Trent may require Bob to provide the message $|P\rangle$ and the corresponding Alice's signature $(|S_A\rangle, R)$. First, Trent checks R is equal to the the random number he saved or not. If R is equal to the the random number he saved, he makes further validation; Otherwise, he concludes the signature is forged by Bob. Trent further verifies whether

$$|S_A\rangle = \mathbb{E}_{K_{AT}}(\mathbb{E}_R(|P\rangle)). \tag{10}$$

If Eq. (10) holds, Trent concludes that $|P\rangle$ is indeed Alice's singed message and Alice is disavowing her signature; Otherwise, he believes the signature is forged by Bob.

5.2 Impossibility of Disavowal by the Signatory and the Receiver

First, we consider the signatory's disavowal in the proposed AQS scheme. If Alice wants to disavow her signature, she must make that Eq. (10) does not hold. However, by Step V2 and Step V7 in the proposed AQS scheme, there are $|S_A\rangle = \mathbb{E}_{K_{AT}}(|P'\rangle)$ and $|P\rangle = \mathbb{E}_R^{-1}(|P'\rangle)$. Therefore, Eq. (10) must hold if Bob receives the quantum message $|Y_T\rangle$ from Trent with no error. Can Alice change $|Y_T\rangle$ when Trent sends it to Bob without being found by Bob?

Due to the unconditionally security of quantum key distribution [18, 20, 24, 25, 27], Alice can not get the key K_{BT} in the initializing phases of the proposed AQS scheme. By Bell states measurement having only four different measurement results, the disavowal attack proposed by Sun *et al.* [28] can only be applied to the AQS schemes [23, 30, 31] which use the QOTP encryption including only four kinds of unitary transformations. Because the SQOTP encryption uses eight kinds of unitary transformations, Alice can not successfully to get K_{BT} by applying the disavowal attack proposed by Sun *et al.* [28] on the proposed AQS scheme. Similarly, the Trojan horse attacks [19] becomes ineffective on the proposed AQS scheme. Furthermore, Trent has reordered all quantum states in $|S_A\rangle$ and $|S_B\rangle$ before he sends them to Bob. So, Alice does not know the exact locations of $|S_A\rangle$ in $|Y_T\rangle$. This is another reason that Alice can not successfully to apply the disavowal attacks proposed by Sun and Hwang *et al.* [19, 28] on the proposed AQS scheme. In addition, Alice can not tamper $|S_A\rangle$ without disturbing $|S_B\rangle$ because she does not know the exact locations of $|S_A\rangle$ in $|Y_T\rangle$. This means that the disavowal attack proposed by Gao *et al.* [17] is invalid on the proposed AQS scheme.

From the above discussion, Alice can not successfully change $|Y_T\rangle$ without being found when Trent sends it to Bob. Therefore, Trent can confirm that Alice has signed the message $|P\rangle$ since Alice's secret key K_{AT} is involved in $|S_A\rangle$ and $|S_A\rangle = \mathbb{E}_{K_{AT}}(\mathbb{E}_R(|P\rangle))$. Hence Alice cannot deny having signed the message.

Now, we show the signature can not be disavowed by the receiver in the proposed AQS scheme. If Bob informs Alice by the public board to publish the random number R in Step V5, this means $|P'_B\rangle = |P'\rangle$ and Bob has obtained $|P'_B\rangle$ and $|P'\rangle$ from $|\mathcal{M}_A\rangle$ and $|Y_T\rangle$. Note that, some one who gets $|P'\rangle$ from $|\mathcal{M}_A\rangle$ must share the Bell states with Alice; and, some one who gets $|P'_B\rangle$ from $|Y_T\rangle$ must share the key K_{BT} with Trent. In addition, Bob can get R by Step V6 because the public board can not be blocked and is assumed not to be the injection or alteration of messages. From the analysis above, Bob can not deny his involving the scheme, or disavow the receipt of $|S\rangle = (|P'\rangle, |S_A\rangle, |\mathcal{M}_A\rangle)$ or the random number R. Therefore, if Bob wants to disavow the signature he must construct a new message $|P_B\rangle$ and a new random number R_B such that $|S_A\rangle = \mathbb{E}_{K_{AT}}(\mathbb{E}_{R_B}(|P_B\rangle))$. However, we will show that Bob can not forge Alice's signature in the following subsection. This means that Bob can not disavow the signature.

Especially, Bob could not claim $|P'_B\rangle \neq |P'\rangle$ when $|P'_B\rangle = |P'\rangle$ if he wants to recover the message $|P\rangle$. If Bob claims $|P'_B\rangle \neq |P'\rangle$, he will not receive the random number R. This means that he will not recover the message $|P\rangle$. As we pointed out in Section 3, it is neither a real deny-integrity attack nor a real DoS attack that Bob refuses to receive the message-signature pair by claiming $|P'_B\rangle \neq |P'\rangle$.

5.3 Impossibility of Forgery

First, we discuss the forgery attack of the receiver Bob. It is easy to know that Bob can not learn the signature key K_{AT} in the initializing phases of the proposed AQS scheme by unconditionally security of quantum key distribution [18, 20, 24, 25, 27]. Furthermore, he can not obtain the signature key K_{AT} in the signing phase and the verifying phase because the SQOTP encryption betrays nothing about the secret keys K_{AT}. Therefore, Bob's the most likely attack is the known message attacks as that in Refs. [11, 17]. However, by Proposition 1, Bob can not find any unitary matrix $U \neq \bigotimes_{i=1}^{n} c_i I$ (I is the second-order identity matrix and c_i a complex number with $|c_i| = 1$) such that $\mathbb{E}_{K_{AT}}(U|P\rangle) = U\mathbb{E}_{K_{AT}}(|P\rangle)$ when he does not know the key K_{AT}. This means that the known message attacks proposed by Gao and Choi *et al.* [11, 17] are ineffective on the proposed AQS scheme. In addition, the forge attack presented by us in Section 3 is also failed on the proposed AQS scheme because Trent has recorded the random number R. Therefore, Bob can not forge Alice's signature.

Now, we consider the forgery attack of outside opponent Eve. Due to the unconditionally security of quantum key distribution [18, 20, 24, 25, 27], she can not get the signature key in the initializing phases of the proposed AQS scheme. In addition, Eve can not acquire the keys K_{AT}, K_{AB} and K_{BT} from the sent quantum messages because the proposed AQS scheme use the SQOTP encryption. This can be proved as the discussion about that Alice can not get the key K_{BT} in the previous subsection. By Proposition 1, Eve can not find any unitary matrix $U \neq \bigotimes_{i=1}^{n} c_i I$ (I is the second-order identity matrix and c_i a complex number with $|c| = 1$) such that $\mathbb{E}_{K_{AT}}(U|P\rangle) = U\mathbb{E}_{K_{AT}}(|P\rangle)$ when she

does not know the key K_{AT}. Furthermore, she does not know the locations of $|S_A\rangle$ in $|Y_T\rangle$ because Trent has reordered $|S_A\rangle$ and $|S_B\rangle$. Therefore, Eve can not forge Alice's signature.

5.4 Other Discussion

It is necessary that Alice only sends $|P'\rangle$ in the proposed AQS scheme. Bob can get the message $|P\rangle$ and confirm $|S_A\rangle$ being Alice's signature in Step V5 if we use the message $|P\rangle$ instead of $|P'\rangle$. Thereby, Bob does not need the random number R. This means that Bob has a chance to disavow the integrality of the signature as the AQS schemes in Refs. [23, 30]. If Bob deliberately declares $|P'_B\rangle \neq |P'\rangle$ in Step V5 of the proposed AQS scheme, he can not obtain the message $|P\rangle$ and its signature because he does not know the the random number R. This only shows that Bob can refuse to receive Alice's message-signature pair, but Bob can not apply the deny-integrity attack. In addition, as we pointed out before, it is not a DoS attack that Bob refuses to receive Alice's message-signature pair by declaring $|P'_B\rangle \neq |P'\rangle$ in Step V5 of the proposed AQS scheme.

Note that, even though Bob and Trent do not use any wavelength filter and PNS, the Trojan horse attacks mentioned in Ref. [19] are invalid on the proposed AQS scheme because it uses the SQOTP encryption. However, we would like to point out that it is important to pay attention to the practical implementation of the AQS schemes.

Finally, we want to point out that the arbitrator can not forge Alice's signature. In the proposed AQS scheme, Alice needs to publish the random number R by the public board. If Trent wants to tamper Alice's signature for some new message, he will not be successful because he does not know the random number R. If Trent impersonates Alice to send a forged message-signature pair $|S\rangle$ to Bob, Alice will found out Trent's impersonation attack in Step V5 because Bob needs Alice announcing her random number.

6 Conclusions

In this paper, we first summarized the existing attacks [11, 17, 28] on AQS schemes, and then presented a valid forgery attack as well as analyzed the reasons why the AQS protocols are susceptible to attacks. In addition, we pointed out that the "deny-integrity attack" in Ref. [19] is not a real deny-integrity attack. Similarly, the "DoS Attack" in Ref. [19] is not a real DoS Attack. To recall the security of the AQS schemes, we constructed an SQOTP encryption. Based on the suggestions in Refs. [11, 17, 28] and our analyses, we proposed an AQS scheme using SQOTP encryption which are secure against all existent attacks [11, 17, 28]. Finally, we showed the security of the proposed AQS scheme. In particular, we showed that the proposed AQS scheme is secure against all existing attacks [11, 17, 28]. The proposed AQS scheme can preserve all merits in the previous AQS schemes [23, 30, 31] such as they can sign the known and unknown quantum messages. Also, they can avoid being disavowed by the receiver. In

addition, the arbitrator can not forge Alice's signature in the proposed AQS scheme. Moreover, they provide higher security than the previous AQS schemes [23, 30, 31] because they can resist all existent attacks [11, 17, 28]. Note that, to achieve the higher security, the proposed AQS scheme need use the SQOTP encryption. This makes the proposed AQS scheme need a longer signature key which is 50% longer than ever.

Acknowledgments. This work is supported in part by the National Natural Science Foundation (Nos. 61272058, 61073054, 60873055), the Natural Science Foundation of Guangdong Province of China (Nos. 10251027501000004, S2012040007324, S2012010008833), the Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 20100171110042), The Foundation for Distinguished Young Talents in Higher Education of Guangdong Province of China (No. 2012LYM_0126), the Science and Technology Project of Jiangmen City of China (No. [2011]131), and the project of SQIG at IT, funded by FCT and EU FEDER projects Qsec PTDC/EIA/67661/2006, AMDSC UTAustin/MAT/0057/2008, NoE Euro-NF, and IT Project QuantTel, FCT project PTDC/EEA-TEL/103402/2008 QuantPrivTel, FCT PEst-OE/EEI/LA0008/2011.

References

1. Aharonov, D., Ben-Or, M.: Fault-tolerant quantum computation with constant error. In: Proceedings of the twenty-ninth Annual ACM Symposium on Theory of Computing, pp. 176–188. ACM (1997)
2. Barnum, H., Crépeau, C., Gottesman, D., Smith, A., Tapp, A.: Authentication of quantum messages. In: Proceedings of the 43rd Annual Symposium on Foundations of Computer Science, pp. 449–458. IEEE (2002)
3. Bennett, C.H.: Quantum cryptography using any two nonorthogonal states. Physical Review Letters 68(21), 3121–3124 (1992)
4. Bennett, C.H., Brassard, G.: Quantum cryptography: Public key distribution and coin tossing. In: Proceedings of IEEE International Conference on Computers, Systems and Signal Processing, pp. 175–179. IEEE (1984)
5. Boyer, M., Gelles, R., Kenigsberg, D., Mor, T.: Semiquantum key distribution. Physical Review A 79(3), 032341 (2009)
6. Boyer, M., Kenigsberg, D., Mor, T.: Quantum key distribution with classical Bob. Physical Review Letters 99(14), 140501 (2007)
7. Boykin, P.O., Roychowdhury, V.: Optimal encryption of quantum bits. Physical Review A 67(4), 042317 (2003)
8. Briegel, H.J., Dür, W., Cirac, J.I., Zoller, P.: Quantum repeaters: The role of imperfect local operations in quantum communication. Physical Review Letters 81(26), 5932–5935 (1998)
9. Buhrman, H., Cleve, R., Watrous, J., De Wolf, R.: Quantum fingerprinting. Physical Review Letters 87(16), 167902 (2001)
10. Cai, Q.Y.: Eavesdropping on the two-way quantum communication protocols with invisible photons. Physics Letters A 351(1), 23–25 (2006)

11. Choi, J.W., Chang, K.Y., Hong, D.: Security problem on arbitrated quantum signature schemes. ArXiv:quant-ph/1106.5318 (2011)
12. Curty, M., Lütkenhaus, N.: Comment on"Arbitrated quantum-signature scheme". Physical Review A 77(4), 046301 (2008)
13. Curty, M., Santos, D.J., Pérez, E., García-Fernández, P.: Qubit authentication. Physical Review A 66(2), 022301 (2002)
14. Deng, F.G., Zhou, P., Li, X.H., Li, C.Y., Zhou, H.Y.: Robustness of two-way quantum communication protocols against trojan horse attack. ArXiv:quant-ph/0508168 (2005)
15. Duan, L.M., Lukin, M.D., Cirac, J.I., Zoller, P.: Generation of nonclassical photon pairs for scalable quantum communication with atomic ensembles. Nature 414, 413 (2001)
16. Ekert, A.K.: Quantum cryptography based on Bell's theorem. Physical Review Letters 67(6), 661–663 (1991)
17. Gao, F., Qin, S.J., Guo, F.Z., Wen, Q.Y.: Cryptanalysis of the arbitrated quantum signature protocols. Physical Review A 84(2), 022344 (2011)
18. Gisin, N., Ribordy, G., Tittel, W., Zbinden, H.: Quantum cryptography. Reviews of Modern Physics 74(1), 145–195 (2002)
19. Hwang, T., Luo, Y.P., Chong, S.K.: Comment on "Security analysis and improvements of arbitrated quantum signature schemes". Physical Review A 85(5), 056301 (2012)
20. Inamori, H., Lütkenhaus, N., Mayers, D.: Unconditional security of practical quantum key distribution. The European Physical Journal D: Atomic, Molecular, Optical and Plasma Physics 41(3), 599–627 (2007)
21. Kwiat, P.G., Mattle, K., Weinfurter, H., Zeilinger, A., Sergienko, A.V., Shih, Y.: New high-intensity source of polarization-entangled photon pairs. Physical Review Letters 75(24), 4337–4341 (1995)
22. Lau, F., Rubin, S.H., Smith, M.H., Trajkovic, L.: Distributed denial of service attacks. In: 2000 IEEE International Conference on Systems, Man, and Cybernetics, vol. 3, pp. 2275–2280. IEEE (2000)
23. Li, Q., Chan, W.H., Long, D.Y.: Arbitrated quantum signature scheme using Bell states. Physical Review A 79(5), 054307 (2009)
24. Lo, H.K., Chau, H.F.: Unconditional security of quantum key distribution over arbitrarily long distances. Science 283(5410), 2050 (1999)
25. Mayers, D.: Unconditional security in quantum cryptography. Journal of the ACM (JACM) 48(3), 351–406 (2001)
26. Shor, P.W.: Fault-tolerant quantum computation. In: Proceedings of the 37th Annual Symposium on Foundations of Computer Science, pp. 56–65. IEEE (1996)
27. Shor, P.W., Preskill, J.: Simple proof of security of the BB84 quantum key distribution protocol. Physical Review Letters 85(2), 441–444 (2000)
28. Sun, Z., Du, R., Long, D.: Improving the security of arbitrated quantum signature protocols. ArXiv:quant-ph/1107.2459 (2011)
29. Zeng, G.: Reply to "Comment on 'Arbitrated quantum-signature scheme' ". Physical Review A 78(1), 016301 (2008)
30. Zeng, G., Keitel, C.H.: Arbitrated quantum-signature scheme. Physical Review A 65(4), 042312 (2002)
31. Zou, X., Qiu, D.: Security analysis and improvements of arbitrated quantum signature schemes. Physical Review A 82(4), 042325 (2010)
32. Zou, X., Qiu, D., Li, L., Wu, L., Li, L.: Semiquantum-key distribution using less than four quantum states. Physical Review A 79(5), 052312 (2009)

Randomized Algorithms for Removable Online Knapsack Problems

Xin Han[1], Yasushi Kawase[2], and Kazuhisa Makino[3]

[1] Dalian University of Technology
hanxin@dlut.edu.cn
[2] University of Tokyo
yasushi_kawase@mist.i.u-tokyo.ac.jp
[3] Kyoto University
makino@kurims.kyoto-u.ac.jp

Abstract. In this paper, we study removable online knapsack problem. The input is a sequence of items $e_1, e_2, \ldots, e_n$, each of which has a weight and a value. Given the ith item e_i, we either put e_i into the knapsack or reject it. When e_i is put into the knapsack, some items in the knapsack are removed with no cost if the sum of the weight of e_i and the total weight in the current knapsack exceeds the capacity of the knapsack. Our goal is to maximize the profit, i.e., the sum of the values of items in the last knapsack. We show a randomized 2-competitive algorithm despite there is no constant competitive deterministic algorithm. We also give a lower bound $1 + 1/e \approx 1.368$. For the unweighted case, i.e., the value of each item is equal to the weight, we propose a 10/7-competitive algorithm and give a lower bound 1.25.

1 Introduction

The knapsack problem is one of the most classical problems in combinatorial optimization and has a lot of applications in the real world [13]. The knapsack problem is that: given a set of items with values and weights, we are asked to maximize the total value of selected items in the knapsack satisfying the capacity constraint. We assume in this paper, the capacity of knapsack is 1.

In this paper, we study the online version of the knapsack problem with removal condition. Here, "online" means i) the information of the input (i.e., the items) is given gradually, i.e., after a decision is made on the current item, the next item is given; ii) the decisions we have made are irrevocable, i.e., once a decision has been made, it cannot be changed. Given the ith item e_i, which has a value $v(e_i)$ and a weight $w(e_i)$, we either accept e_i (i.e., put e_i into the knapsack) or reject it. When e_i is put into the knapsack, some items in the knapsack are removed with no cost if the sum of the weight of e_i and the total weight in the current knapsack exceeds 1, i.e., the capacity of the knapsack. Our goal is to maximize the profit, i.e., the sum of the values of items in the last knapsack.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 60–71, 2013.

Related Works

It is well known that offline knapsack problem is NP-hard but has an FPTAS as well as a simple 2-approximation. Kellerer *et al.* [12] gave a 5/4-approximation linear time algorithm for unweighted case, i.e., the value of each item is equal to the weight. Ito *et al.* [9] gave a constant-time approximation algorithm using weighted sampling.

An online knapsack problem was first studied on average case analysis by Marchetti-Spaccamela and Vercellis [15]. They proposed a linear time approximation algorithm such that the expected difference between the optimal profit and the one obtained by the algorithm is $O(\log^{3/2} n)$ under the condition that the capacity of the knapsack grows proportionally to the number of items n. Lueker [14] improved the expected difference to $O(\log n)$ under a fairly general condition on the distribution.

On worst case analysis, Buchbinder and Naor [4,5] gave an $O(\log(U/L))$-competitive algorithm based on a general online primal-dual framework when the density (value to weight ratio) of every element is in a known range $[L, U]$, and that each weight is much smaller than the capacity of the knapsack. They also showed an $\Omega(\log(U/L))$ lower bound on the competitive ratio for the case.

Iwama and Taketomi [10] studied *removable* online knapsack problem. They obtained a $(1 + \sqrt{5})/2 \approx 1.618$-competitive algorithm for the online knapsack when (i) the removable condition is allowed and (ii) the value of each item is equal to the weight (unweighted), and showed that this is best possible by providing a lower bound 1.618 for the case. We remark that the problem has unbounded competitive ratio, if at least one of the conditions (i) and (ii) is not satisfied [10,11]. For randomized case, Babaioff *et al.* [3] showed a lower bound 5/4 for the weighted case.

Removable online knapsack problem with cancellation cost is studied in [2,3,6]. When the cancellation cost is f times the total value of removed items, Babaioff *et al.* [2,3] showed that if the largest element is of size at most γ, where $0 < \gamma < 1/2$, then the competitive ratio is $1 + 2f + 2\sqrt{f(1+f)}$ with respect to the optimum solution for the knapsack problem with capacity $(1 - 2\gamma)$. They also showed randomized $3(1 + 2f + 2\sqrt{f(1+f)})$-competitive algorithm for this problem. Han *et al.* [6] studied the problem of unweighted case. They proved the problem is $\max\{2, \frac{1+f+\sqrt{f^2+2f+5}}{2}\}$ competitive.

For other models such as knapsack secretary problem and online minimization knapsack problem, refer to papers in [1,7,8].

Our Results

In this paper, we study the worst case analysis of randomized removable online knapsack problem (against an oblivious adversary). For unweighted case, we give a randomized 10/7-competitive algorithm. The main ideas of our algorithm are below: we divide all the items into three groups, *small*, *median* and *large*. If there is a large item, we just select and cancel all we have selected, otherwise we first handle median items, then apply a greedy algorithm for the small items.

For median items, we have two subroutines and run each of them with probability 0.5, respectively. We also show there is no randomized online algorithm with competitive ratio less than 5/4 for the unweighted case.

For weighted case, we give a randomized 2-competitive algorithm. This is an extension of 2-approximation greedy algorithm for offline knapsack problem. We also show there is no randomized online algorithm with competitive ratio less than $1 + 1/e$ for the online removable weighted knapsack problem.

We summarize competitive ratios for removable knapsack problems in Table 1, where our results are written in bold letters.

The rest of paper is organized as follows. In Section 2, we consider the unweighted case, and in Section 3, we consider the weighted case.

Table 1. Competitive ratios for removable knapsack problems, where our results are written in bold letters

	unweighted		weighted	
	upper bound	lower bound	upper bound	lower bound
deterministic	$\frac{1+\sqrt{5}}{2}$ ($\approx$1.618) [10]	$\frac{1+\sqrt{5}}{2}$ [10]	∞ [11]	∞ [11]
randomized	**10/7** (**$\approx$1.429**)	**5/4** (**=1.25**)	**2**	**$1 + \frac{1}{e}$** (**$\approx$1.368**)

2 Unweighted Knapsack Problem

In this section, we consider removable knapsack problem when the value of each item is equal to the weight.

2.1 Upper Bound

In this subsection, we propose a randomized 10/7-competitive online algorithm for unweighted removable knapsack problem.

We divide all the items into three groups, *small*, *median* and *large*, for short, say S, M, L. Given an item with size x, if $x \leq 0.3$, we call the item small, if $0.3 < x < 0.7$, we call it median, otherwise (i.e., $x \geq 0.7$) we call it large. Further, we divide median items into four subgroups, M_i for $1 \leq i \leq 4$. Given an item of size x, if $x \in (0.3, 0.4]$, then it belongs to M_1; if $x \in (0.4, 0.5]$, then it belongs to M_2; if $x \in (0.5, 0.6)$, then it belongs to M_3; if $x \in [0.6, 0.7)$, then it belongs to M_4 (see Fig. 1).

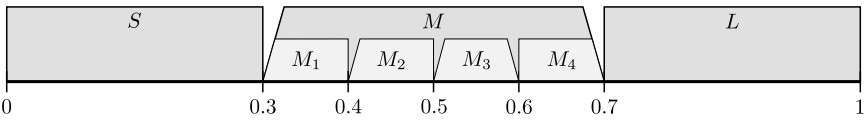


Fig. 1. Groups of items

The main ideas of our algorithm are below: if there is a large item, we just select it and cancel all we have selected, otherwise we first handle median items, then apply a greedy algorithm for the small items. For median items, we have two subroutines and run each of them with probability 0.5, respectively.

In our algorithm, greedy algorithm G_1 means that we select items by a way from the large to the small; greedy algorithm G_2 means that we select items by a way from the small to the large. For example, assume that the input sequence is $(0.42, 0.3, 0.2, 0.11, 0.09, 0.6)$. Then at each round, the solution by G_1 is (0.42), $(0.42, 0.3)$, $(0.42, 0.3, 0.2)$, $(0.42, 0.3, 0.2)$, $(0.42, 0.3, 0.2)$ and $(0.6, 0.3)$, respectively. And at each round, the solution by G_2 is (0.42), $(0.3, 0.42)$, $(0.2, 0.3, 0.42)$, $(0.11, 0.2, 0.3)$, $(0.09, 0.11, 0.2, 0.3)$ and $(0.09, 0.11, 0.2, 0.3)$, respectively. There is a flag called f in our algorithm, where $f = 1$ means there is an item with size in $[0.6, 0.7)$, otherwise $f = 0$. Let e_i be the item given in the ith round. Define by B_i the set of selected items at the end of the ith round by Algorithm A. For $r = 1, 2$, define by B_i^r the set of selected items at the end of the ith round by Algorithm A_r.

Algorithm A

$B_0,\ B_0^1,\ B_0^2 \leftarrow \emptyset,\ f \leftarrow 0$
choose r at random from $\{1, 2\}$
for all items e_i, in order of arrival, **do**
  if $e_i \geq 0.7$ **then** select it, cancel all the items selected in the knapsack
  else if $f = 0$ and $e_i \in [0.6, 0.7)$ **then** $f \leftarrow 1$
  simulate two algorithms $A_1(f, e_i)$ and $A_2(f, e_i)$;
  if $r = 1$ **then** $B_i \leftarrow B_i^1$
  if $r = 2$ **then** $B_i \leftarrow B_i^2$
  if the expected profit by our algorithm $(B_i^1 + B_i^2)/2$ is at least 0.7, **then** stop handling the future items.
end for

The main ideas of the following subroutine A_1 are if there are M_4 items, we select the minimum item of them; else if there are M_3 items, we select the minimum item of them; finally we run G_1 algorithm for selecting items from the large to the small.

Subroutine A_1

if $f = 0$ **then** select the minimum item in $(0.5, 0.6)$ from $B_{i-1}^1 \cup \{e_i\}$;
else $f = 1$: select the minimum item in $[0.6, 0.7)$ from $B_{i-1}^1 \cup \{e_i\}$.
run G_1 on the other items from $B_{i-1}^1 \cup \{e_i\}$.

The main ideas of the following subroutine A_2 are: in all the items we are holding (including the new item), if there is a feasible solution with profit at least 0.9 then we select that feasible solution; else if there is an M_4 item and a feasible solution with profit at least 0.8 then we select that feasible solution; else we select the minimum M_2 and the minimum M_1 first if possible, then run algorithm G_2 for selecting all the other median items from the small to the large, finally run G_1 algorithm for selecting small items from the large to the small.

Subroutine A_2

if $f = 0$ and there is a feasible solution with profit at least 0.9 in $B^2_{i-1} \cup \{e_i\}$
  then get that solution.
else if $f = 1$ and there is a feasible solution with profit at least 0.8 in $B^2_{i-1} \cup \{e_i\}$
  then get that solution.
else
  select the minimum item in $(0.4, 0.5]$ from $B^2_{i-1} \cup \{e_i\}$ first;
  then select the minimum item in $(0.3, 0.4]$ from $B^2_{i-1} \cup \{e_i\}$;
  then run G_2 for all the other median items in $B^2_{i-1} \cup \{e_i\}$,
  i.e, the smaller one has a higher priority;
  finally run G_1 for the small items.

Let T be an input sequence, and let m_i be the minimum M_i-item in T. We denote by $A_1(T)$, $A_2(T)$, and $\mathbf{E}[A(T)]$, the profit by A_1, A_2, and the expected profit by A for the input sequence T.

Observation 1. *Given an input T, if one of the following conditions holds, then $\mathbf{E}[A(T)] \geq 0.7$: i) there is a large item in T; ii) $f = 1$ and $A_2(T) \geq 0.8$; iii) $f = 0$ and $A_2(T) \geq 0.9$; iv) $A_1(T) \geq 0.7$ and $A_2(T) \geq 0.7$.*

Observation 2. *If $\mathbf{E}[A(T)] < 0.7$ then items m_1 and m_2 must be selected by algorithm A_2 if they exist.*

Lemma 1. *If $\mathbf{E}[A(T)] < 0.7$, then an M_3-item and an M_1-item cannot co-exist in T.*

Proof. Since $\mathbf{E}[A(T)] < 0.7$, all the items in T are at most 0.7. Assume there are some M_1-items and M_3-items in T. By the definitions of M_1 and M_3, any pair of an M_1-item and an M_3-item can be put together in the knapsack. Let item f_i be the first M_i item in T. Next we prove that $\mathbf{E}[A(T)] > 0.7$, if items f_1 and f_3 exists in T. There are two cases.

Case 1: the first M_3-item f_3 arrives earlier than the first M_1-item f_1.

Case 1.1: item f_4 arrives earlier than item f_1. Before item f_1 arrives, a smallest M_4-item m_4 must be selected by A_1. After item f_1 is given, algorithm A_2 will select item f_1 and an M_2-item, or item f_1 and an M_3-item, i.e., $A_2(T) > 0.4 + f_1$. When item f_1 is given, if $f_1 + m_4 \leq 1$, then items m_4 and f_1 will be selected by A_1, i.e., $A_1(T) > 0.6 + 0.3 = 0.9$. We are done since $\mathbf{E}[A(T)] = \frac{A_1(T)+A_2(T)}{2} > \frac{0.9+0.4+0.3}{2} > 0.8$. Otherwise item m_4 rejects item f_1, i.e., $f_1 + m_4 > 1$. Thus we have $\mathbf{E}[A(T)] = \frac{A_1(T)+A_2(T)}{2} > \frac{f_1+m_4+0.4}{2} > 0.7$, where $f_1 + m_4 > 1$.

Case 1.2: there is no M_4-item before item f_1. Then just after item f_1, algorithm A_1 will select an M_3-item, and item f_1 or an M_2-item, i.e., $A_1(T) > 0.5 + 0.3 = 0.8$. Algorithm A_2 will select item f_1, and an M_2 or M_3-item, i.e., $A_2(T) > 0.4 + 0.3 = 0.7$, thus we have $\mathbf{E}[A(T)] = \frac{A_1(T)+A_2(T)}{2} > \frac{0.8+0.7}{2} = 0.75$.

Case 2: the first M_1-item f_1 arrives earlier than the first M_3-item f_3.

Case 2.1: item f_4 arrives earlier than item f_3. After item f_3 arrives, item f_4 must be selected by A_1, and an M_1-item must be selected by A_2. Since item f_3

and an M_1-item forms a feasible solution with profit at least 0.8 and $f = 1$, we have $\mathbf{E}[A(T)] = \frac{A_1(T)+A_2(T)}{2} > \frac{0.6+0.8}{2} = 0.7$.

Case 2.2: there is no M_4-item before item f_3. Just before item f_3 arrives, if there are two M_2-items accepted by algorithm A_1, then $\mathbf{E}[A(T)] > (0.8+0.7)/2 = 0.75$ since an M_1-item and an M_2-item will be selected by A_2. Thus, just before item f_3 arrives, there is at most one M_2-item in T, and at least one M_1-item is selected by A_1. Just after item f_3 arrives, algorithm A_1 will select item f_3, and an M_2-item or an M_1-item, i.e., $A_1(T) > 0.5 + 0.3 = 0.8$. Algorithm A_2 will select an M_1-item, and an M_2-item or item f_3, i.e., $A_2(T) > 0.3 + 0.4 = 0.7$. Thus we have $\mathbf{E}[A(T)] = \frac{A_1(T)+A_2(T)}{2} > \frac{0.8+0.7}{2} = 0.75$.

In both cases, we have $\mathbf{E}[A(T)] > 0.7$, which causes a contradiction with $\mathbf{E}[A(T)] < 0.7$. Hence this lemma holds. □

Lemma 2. *If* $\mathbf{E}[A(T)] < 0.7$ *and there are some* M_1*-items and* M_4*-items in* T*, then we have* $m_1 + m_4 > 1$*.*

Proof. Assume $m_1 + m_4 \leq 1$. Next we prove that after handling two items m_1 and m_4, the expected profit by A would be larger than 0.7. There are two cases.

Case 1: item m_1 arrives earlier than item m_4. After item m_4 is given, it must be selected by algorithm A_1, and we have $f = 1$. We also know that m_1 must be selected in the knapsack by A_2. Since $f = 1$ and items m_1 and m_4 form a feasible solution with profit larger than 0.9, we have $\mathbf{E}[A(T)] = \frac{A_1(T)+A_2(T)}{2} > \frac{0.6+0.9}{2} = 0.75$.

Case 2: item m_4 arrives earlier than item m_1. After item m_1 is given, A_1 must select m_4 and an M_1-item, i.e., $A_1(T) > 0.9$. Observe that by Lemma 1, there is no M_3-item before m_1. If there is an M_1 or M_2-item before item m_1, then two items in $(0.3, 0.5]$ must be selected by A_2 after m_1, otherwise items m_4 and m_1 will be selected. Thus A_2 must select two median items after m_1. Then we have $\mathbf{E}[A(T)] = \frac{A_1(T)+A_2(T)}{2} > \frac{0.9+0.6}{2} = 0.75$. Hence the assumption is wrong, we have $m_1 + m_4 > 1$. □

Lemma 3. *If* $\mathbf{E}[A(T)] < 0.7$ *and* $f = 1$*, and there are some* M_2*-items and* M_3*-items in* T*, then we have* $m_2 + m_3 > 1$*.*

Proof. Assume $m_2 + m_3 \leq 1$. By Lemma 1 there is no M_1-item in T. Next we prove that after handling two items m_2 and m_3, the expected profit by A would be larger than 0.7. Let item f_4 be the first M_4 item in T. If item f_4 arrives later than items m_2 and m_3, then algorithm A_1 selects f_4. Algorithm A_2 selects item m_2, and another M_2-item or item m_3 just after item f_4 arrives. Thus we have $\mathbf{E}[A(T)] > (0.6 + 2 \cdot 0.4)/2 = 0.7$. Next we consider the case: item f_4 arrives earlier than items m_2 or m_3. There are two cases.

Case 1: items m_2 and f_4 arrive earlier than item m_3. After item m_3 arrives, algorithm A_1 selects at least one item f_4. Since there is no M_1-item in T, and items m_2 and m_3 or two M_2-items form a feasible solution with profit larger than 0.8, algorithm A_2 selects item m_2, and another M_2-item or item m_3. thus we have $\mathbf{E}[A(T)] > (0.6 + 0.8)/2 = 0.7$.

Case 2: items m_3 and f_4 arrive earlier than item m_2. After item m_2 arrives, algorithm A_1 selects at least one item f_4. Since there is no M_1-item in T, and items m_2 and m_3 or two M_2-items form a feasible solution with profit larger than 0.8, algorithm A_2 selects at least two items larger than 0.4. Thus we have $\mathbf{E}[A(T)] > (0.6 + 0.8)/2 = 0.7$.

Hence the assumption is wrong, we have $m_2 + m_3 > 1$. □

Lemma 4. *If* $\mathbf{E}[A(T)] < 0.7$*, there is at most one* M_2*-item in* T*.*

Proof. Assume there are two M_2-items in T, say item a and b. Without loss of generality, assume item a arrives earlier than item b. Let f_i be the first M_i-item in T. There are three cases.

Case 1: item b arrives earlier than items f_3 and f_4. After item b arrives, algorithm A_1 selects two M_2-items and algorithm A_2 selects two M_2-items, or an M_1-item and an M_2-item. Thus we have $\mathbf{E}[A(T)] > (2 \cdot 0.4 + 0.3 + 0.4)/2 = 0.75$.

Case 2: item b arrives later than f_4. After item b arrives, algorithm A_1 selects at least one item f_4. Since $f = 1$ and two M_2-items form a feasible solution with profit larger than 0.8, $A_2(T) > 0.8$. Thus we have $\mathbf{E}[A(T)] > (0.6 + 0.8)/2 = 0.7$.

Case 3: item b arrives later than f_3. After item b arrives, algorithm A_1 selects at least one item f_3. Algorithm A_2 selects two M_2-items since by Lemma 1 there is no M_1-item in T. By Lemma 3, we have $f_3 + m_2 > 1$, thus we have $\mathbf{E}[A(T)] > (f_3 + m_2 + 0.4)/2 > 0.7$.

Hence the assumption is wrong, there is at most one M_2-item in T. □

Let $OPT(T)$ be an optimal offline value for T.

Lemma 5. *If there is no small item in* T*, then* $OPT(T)/\mathbf{E}[A(T)] \leq 10/7$*.*

Proof. Assume that online algorithm A does not stop handling a future item, i.e., we have $\mathbf{E}[A(T)] < 0.7$ for an input T. By Observation 1, if $A_2(T) \geq 0.9$, we have $\mathbf{E}[A(T)] \geq 0.7$. Hence we assume $A_2(T) < 0.9$.

Let $|OPT|$ be the number of M-items in an optimal solution. We first claim that $|OPT| \leq 2$. Since every item has size at least 0.3, we have $|OPT| \leq 3$. If $|OPT| = 3$, then the three items must be M_1-items. If there were no M_2-item in T, then algorithm A_2 would have selected three items, i.e., $A_2(T) > 0.9$, and $\mathbf{E}[A(T)] > 0.7$. Hence algorithm A_2 selects two items and one of them must be M_2-item, i.e., $A_2(T) = m_1 + m_2$. If $f = 1$ then we have $\mathbf{E}[A(T)] = \frac{A_1(T)+A_2(T)}{2} \geq \frac{m_4+m_1+m_2}{2} > \frac{1+m_2}{2} \geq 0.7$, where $m_1 + m_4 > 1$ holds by Lemma 2. Otherwise $f = 0$. By Lemma 1, there is no M_3-item in T. According to algorithm A_1, the largest two items in T must be selected by A_1, then $A_1(T) \geq m_1 + m_2$. Hence $\mathbf{E}[A(T)] = \frac{A_1(T)+A_2(T)}{2} \geq m_1 + m_2 > 0.7$. In each case, we have $\mathbf{E}[A(T)] > 0.7$, which causes a contradiction with the assumption $\mathbf{E}[A(T)] < 0.7$. Thus we have $|OPT| \leq 2$.

Case 1: $|OPT| = 1$, i.e., $OPT(T) < 0.7$. If $0.5 < OPT(T)$, i.e., there is an M_3 or M_4-item in T, then item m_3 or m_4 must be selected by algorithm A_1.

And A_2 at least selects one item a. We claim that $a + m_3 > 1$ or $a + m_4 > 1$, otherwise $OPT(T) > 0.7$. Hence we have $\mathbf{E}[A(T)] = \frac{A_1(T)+A_2(T)}{2} > \frac{1}{2} \geq \frac{OPT(T)}{1.4}$. Otherwise $OPT(T) \leq 0.5$, i.e., there is only one item in T, otherwise $OPT(T)$ would be larger than 0.5. Then we have $A_1(T) = A_2(T) = OPT(T)$, hence $\mathbf{E}[A(T)] = (A_1(T) + A_2(T))/2 = OPT(T)$.

Case 2: $|OPT| = 2$. let items a and b are the two items, where $a \geq b$. We claim that $a \leq 0.5$. If item a is an M_4-item, by Lemma 2, we have $a + b > 1$. If item a is an M_3-item, by Lemma 3, item b must be an M_1-item since $a + b \leq 1$. On the other hand, if item a is an M_3-item, by Lemma 1, item b cannot be an M_1-item. Thus we have $a \leq 0.5$ and items a and b are the largest two items in T. According to algorithm A_1, items a and b must be selected by A_1, i.e., $A_1(T) = OPT(T)$. On the other hand, algorithm A_2 must select at least two items, i.e., $A_2(T) > 0.6$. Hence $\mathbf{E}[A(T)] = \frac{A_2(T)+A_1(T)}{2} = \frac{OPT(T)+0.6}{2} \geq 0.8 \cdot OPT(T)$, where $OPT(T) \leq 1$.

Hence $OPT(T)/\mathbf{E}[A(T)] < 10/7$. □

Lemma 6. *Even if there are some small items in T, we have $OPT(T)/\mathbf{E}[A(T)] \leq 10/7$.*

Proof. Assume $\mathbf{E}[A(T)] < 0.7$, otherwise we have $OPT(T)/\mathbf{E}[A(T)] \leq 10/7$. If algorithms A_1 and A_2 accept all the small items in T, we have $OPT(T)/\mathbf{E}[A(T)] \leq 10/7$ by Lemma 5. Assume item a is the first small item which is rejected by A_1 or A_2 algorithm. If both algorithms reject a small item, then we have $\min\{A_1(T), A_2(T)\} \geq 0.7$, then $\mathbf{E}[A(T)] \geq 0.7$. And, if there is no median item in T, we have $OPT(T)/\mathbf{E}[A(T)] \leq 10/7$.

Let T^m be the sub-sequence of T after removing all the small items. If $\min\{A_1(T^m), A_2(T^m)\} \geq 0.4$, we have $A_1(T) + A_2(T) \geq 1.4$. The reason is that: for $i, i' \in \{1, 2\}$ $(i \neq i')$, item a was rejected by A_i, i.e., $a + A_i(T) > 1$, item a was accepted by $A_{i'}$, i.e., $A_{i'}(T) \geq a + A_{i'}(T^m) \geq a + 0.4$. Therefore $A_1(T) + A_2(T) \geq a + 0.4 + A_i(T) > 1.4$. Otherwise there are two cases.

Case 1: $0 < A_1(T^m) < 0.4$, i.e., there is only one median item in T. Two algorithms A_1 and A_2 will perform exactly the same operations, which causes a contradiction with the fact one rejects a small item and the other one does not.

Case 2: $0 < A_2(T^m) < 0.4$, i.e., there is one M_1-item in T, and no M_2 or M_3-item. We claim that there is an M_4-item in T, otherwise two algorithms A_1 and A_2 will perform exactly the same operations. Thus we have $A_1(T^m) \geq 0.6$. We also know algorithm A_1 rejects item a. Since $A_1(T^m) < 0.7$, A_1 accepts at least one small item b, which is not smaller than item a by the greedy algorithm used by A_1, i.e., $b \geq a$. By Lemma 2, $m_1 + m_4 > 1$. The total of small items accepted by A_1 is at least $\min\{0.3 - a, b\} \geq \min\{0.3 - a, a\} \geq 0.15$ and the total of small items accepted by A_2 is $\min\{a + b, 0.3\} \geq \min\{0.3, 2a\} \geq 0.3$. Then we have $\mathbf{E}[A(T)] \geq \frac{A_1(T)+A_2(T)}{2} \geq \frac{m_4+0.15+m_1+0.3}{2} = 0.725 \geq 0.725 \cdot OPT(T)$. Hence we have this lemma. □

By Lemmas 5 and 6, we have the following theorem.

Theorem 1. *The algorithm A is $10/7$-competitive for the online removable unweighted knapsack problem.*

Before concluding this subsection, we remark that the condition in the first and third line of Subroutine A_2 can be checked efficiently.

Proposition 1. *The conditions in the first and third lines of Subroutine A_2 can be checked in linear time.*

Proof. Let B be $B^2_{i-1} \cup \{e_i\}$, where B^2_{i-1} is the set of items selected by Subroutine A_2 and e_i is the item arrived at the ith round. Our goal is to decide whether there is $B' \subseteq B$ such that $t \leq B^{'} \leq 1$ for $t = 0.8$ or 0.9 in $O(|B|)$ time.

Let $B = \{b_1, b_2, \ldots, b_m\}$ satisfy $b_1 \geq \cdots \geq b_k > 1 - t \geq b_{k+1} \geq \cdots \geq b_m$. It is not difficult to see that $k \leq 10$, since the total size of items in B^2_{i-1} is at most 1 and $b_i > 0.1$ for $1 \leq i \leq k$. Let B_l be $\{b_1, b_2, \ldots, b_k\}$. Let B_s be $\{b_{k+1}, \ldots, b_m\}$. We check whether there is a feasible solution in B with profit at least t by the following way: for each subset B' of items in B_l, we add items in B_s to B' one by one in an arbitrary order. Observe if there is some item left in B_s, then the total size of B' is at least t. If we find a set of items B' with profit at least t, then we are done, otherwise there is not such feasible solution. The total number of subsets B' is at most $2^k \leq 1024$. Thus the conditions in the first and third lines can be check in linear time. □

2.2 Lower Bound

Babaioff *et al.* [3] showed a lower bound $5/4$ for the weighted case, however, we show it for the unweighted case. The proof is based on Yao's Principle [16]. We consider the following input distribution:

$$\begin{cases} 2/3+\varepsilon,\ 1/3,\ 2/3 & \text{(with probability } 1/2\text{)}, \\ 2/3+\varepsilon,\ 1/3, & \text{(with probability } 1/2\text{)} \end{cases} \tag{1}$$

where ε is a sufficiently small positive number.

Theorem 2. *There is no randomized online algorithm with competitive ratio less than $5/4$ for the online removable unweighted knapsack problem.*

Proof. We consider the input distribution (1). Then, the optimal expected profit is $1 \cdot \frac{1}{2} + \frac{2}{3} \cdot \frac{1}{2} = \frac{5}{6}$.

Let A be an online deterministic algorithm chosen arbitrarily. If A rejects the second item, the expected profit is at most $2/3+\varepsilon$. Otherwise, A takes the second item (and removes the first item), the expected profit is at most $1 \cdot \frac{1}{2} + \frac{1}{3} \cdot \frac{1}{2} = \frac{2}{3}$.

Therefore, the competitive ratio is at least $(5/6)/(2/3+\varepsilon)$ which approaches to $5/4$ as $\varepsilon \to 0$.

□

3 Weighted Knapsack Problem

In this section, we consider removable knapsack problem in general case.

3.1 Upper Bound

Nevertheless there is no deterministic online algorithm with constant competitive ratio [11], we propose a 2-competitive randomized algorithm. We use two algorithms *MAX* and *GREEDY*, as an extension of 2-approximation algorithm for offline knapsack problem.

Let e_i be the item given in the ith round. Define by B_i the set of selected items at the end of the ith round, and by $w(B_i)$, $v(B_i)$ the total weight and value in B_i, respectively.

Algorithm *MAX*

$B_0 \leftarrow \emptyset$
for all items e_i, in order of arrival, **do**
  $B_i \leftarrow \text{argmax}\{v(B) : B \subseteq B_{i-1} \cup \{e_i\},\ w(B) \leq 1\}$
end for

Algorithm *GREEDY*

$B_0 \leftarrow \emptyset$
for all items e_i, in order of arrival, **do**
  Let $B_{i-1} \cup \{e_i\} = \{b_1, \ldots, b_k\}$ s.t. $\frac{v(b_1)}{w(b_1)} \geq \frac{v(b_2)}{w(b_2)} \geq \cdots \geq \frac{v(b_k)}{w(b_k)}$
  $B_i \leftarrow \emptyset$
  for $j = 1$ to k **do**
    if $w(B_i) + w(b_j) \leq 1$ **then** $B_i \leftarrow B_i \cup \{b_j\}$
  end for
end for

Let $T = \{e_1, e_2, \ldots, e_n\}$ be an input sequence, $OPT(T)$ be the optimal offline solution for T.

Theorem 3. *Running Algorithms MAX and GREEDY uniformly at random is* 2*-competitive.*

Proof. For the input sequence T, we denote by $MAX(T)$, $GREEDY(T)$, the profit by algorithms *MAX*, *GREEDY*, respectively. By the definitions of Algorithms *MAX* and *GREEDY*, we have $OPT(T) \leq MAX(T) + GREEDY(T)$, since the optimal fractional value of Knapsack problem is not smaller than the optimal integral value of the problem and the fractional solution is above bounded by $GREEDY(T) + MAX(T)$. Therefore, the competitive ratio is at most $\frac{2OPT(T)}{MAX(T)+GREEDY(T)} \leq 2$. □

3.2 Lower Bound

We first show, the competitive ratio 2 is tight for *MAX* and *GREEDY* at the same time.

Proposition 2. *For any positive number* ε*, there is an input sequence such that both Algorithms MAX and GREEDY are at least* $(2 - \varepsilon)$*-competitive simultaneously.*

Proof. Let (w, v) denote an item whose weight and value are w and v, respectively. Consider the sequence of items

$$(1,1),\ \left(\frac{1}{2}+\frac{1}{2n},\ 1-\frac{1}{2n}\right),\ \underbrace{\left(\frac{1}{2n},\frac{1}{n}\right),\ \left(\frac{1}{2n},\frac{1}{n}\right),\ldots,\ \left(\frac{1}{2n},\frac{1}{n}\right)}_{n \text{ items}}$$

where n is a natural number greater than $3/\varepsilon$. Then the optimal offline algorithm gets a value of $(2-3/n)$ by picking the second item and $n-1$ of items with $(\frac{1}{2n}, \frac{1}{n})$. The profit of Algorithm *MAX* is 1 by picking the first item. The profit of Algorithm *GREEDY* is also 1 by picking the all of items with $(\frac{1}{2n}, \frac{1}{n})$. Therefore, the competitive ratio is at least $\frac{2-3/n}{1} \geq 2-\varepsilon$ for both algorithms. □

Next we prove the lower bound $1+1/e$ on the competitive ratio of weighted case using Yao's Principle [16]. We give a family of input distributions parametrized by a natural number n. Let (w, v) denote an item whose weight and value are w and v, respectively. For a given n, the input sequence is

$$(1,1),\ \underbrace{(1/n^2,1/n),\ (1/n^2,1/n),\ldots,\ (1/n^2,1/n)}_{k \text{ items}} \tag{2}$$

with probability $p_k = \frac{1-e^{-1/n}}{1-e^{-n}} \cdot e^{-(k-1)/n}$ for $k = 1, 2, \ldots, n^2$.

Theorem 4. *There is no randomized online algorithm with competitive ratio less than* $1+1/e$ *for the online removable weighted knapsack problem.*

Proof. We consider the input distribution (2). Then, the optimal expected profit is

$$\sum_{i=1}^{n} 1 \cdot p_i + \sum_{i=n+1}^{n^2} \frac{i}{n} \cdot p_i = \frac{1-e^{-1/n}}{1-e^{-n}} \left(\sum_{i=1}^{n} e^{-(i-1)/n} + \frac{1}{n} \sum_{i=n+1}^{n^2} i \cdot e^{-(i-1)/n} \right)$$
$$\to \int_0^1 e^{-t}\,dt + \int_1^\infty t \cdot e^{-t}\,dt = 1 + \frac{1}{e} \quad (n \to \infty).$$

For an online deterministic algorithm A chosen arbitrarily, let $l = n \cdot x$ be the number of input items that A rejects. Then, the expected profit of the algorithm A is at most

$$\sum_{i=1}^{l} p_i + \sum_{i=l+1}^{n^2} \frac{i-l}{n} \cdot p_i = \frac{1-e^{-1/n}}{1-e^{-n}} \left(\sum_{i=1}^{l} e^{-(i-1)/n} + \frac{1}{n} \sum_{i=l+1}^{n^2} (i-l) \cdot e^{-(i-1)/n} \right)$$
$$\to \int_0^x e^{-t}\,dt + \int_x^\infty (t-x) \cdot e^{-t}\,dt = 1 \quad (n \to \infty).$$

Therefore the competitive ratio for the unweighted case is at least $1+1/e$ for any online algorithm. □

Acknowledgment. This work is partially supported by NSFC(11101065) and "the Fundamental Research Funds for the Central Universities" (DUT12LK09), by Grant-in-Aid for Scientific research from the Ministry of Education, Culture, Sports, Science and Technology of Japan, and by the Global COE "The Research and Training Center for New Development in Mathematics", and by JST, ERATO, Kawarabayashi Large Graph Project.

References

1. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: A knapsack secretary problem with applications. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 16–28. Springer, Heidelberg (2007)
2. Babaioff, M., Hartline, J.D., Kleinberg, R.D.: Selling banner ads: Online algorithms with buyback. In: Proceedings of 4th Workshop on Ad Auctions (2008)
3. Babaioff, M., Hartline, J.D., Kleinberg, R.D.: Selling ad campaigns: Online algorithms with cancellations. In: ACM Conference on Electronic Commerce, pp. 61–70 (2009)
4. Buchbinder, N., Naor, J(S.): Online primal-dual algorithms for covering and packing problems. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 689–701. Springer, Heidelberg (2005)
5. Buchbinder, N., Naor, J.: Improved bounds for online routing and packing via a primal-dual approach. In: Foundations of Computer Science, pp. 293–304 (2006)
6. Han, X., Kawase, Y., Makino, K.: Online knapsack problem with removal cost. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 61–73. Springer, Heidelberg (2012)
7. Han, X., Makino, K.: Online minimization knapsack problem. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 182–193. Springer, Heidelberg (2010)
8. Han, X., Makino, K.: Online removable knapsack with limited cuts. Theoretical Computer Science 411, 3956–3964 (2010)
9. Ito, H., Kiyoshima, S., Yoshida, Y.: Constant-time approximation algorithms for the knapsack problem. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 131–142. Springer, Heidelberg (2012)
10. Iwama, K., Taketomi, S.: Removable online knapsack problems. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 293–305. Springer, Heidelberg (2002)
11. Iwama, K., Zhang, G.: Optimal resource augmentations for online knapsack. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 180–188. Springer, Heidelberg (2007)
12. Kellerer, H., Mansini, R., Speranza, M.G.: Two linear approximation algorithms for the subset-sum problem. European Journal of Operational Research 120, 289–296 (2000)
13. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
14. Lueker, G.S.: Average-case analysis of off-line and on-line knapsack problems. Journal of Algorithms 29(2), 277–305 (1998)
15. Marchetti-Spaccamela, A., Vercellis, C.: Stochastic on-line knapsack problems. Mathematical Programming 68, 73–104 (1995)
16. Yao, A.: Probabilistic computations: Toward a unified measure of complexity. In: 18th Annual Symposium on Foundations of Computer Science, pp. 222–227. IEEE (1977)

An Exact Algorithm for Maximum Independent Set in Degree-5 Graphs

Mingyu Xiao[1] and Hiroshi Nagamochi[2]

[1] School of Computer Science and Engineering,
University of Electronic Science and Technology of China, China
myxiao@gmail.com
[2] Department of Applied Mathematics and Physics,
Graduate School of Informatics, Kyoto University, Japan
nag@amp.i.kyoto-u.ac.jp

Abstract. The maximum independent set problem is a basic NP-hard problem and has been extensively studied in exact algorithms. The maximum independent set problems in low-degree graphs are also important and may be bottlenecks of the problem in general graphs. In this paper, we present an $O^*(1.1737^n)$-time exact algorithm for the maximum independent set problem in an n-vertex graph with degree bounded by 5, improving the previous running time bound of $O^*(1.1895^n)$. In our algorithm, we introduce an effective divide-and-conquer procedure to deal with vertex cuts of size at most two in graphs, and design branching rules on some special structures of triconnected graphs of maximum degree 5. These result in an improved algorithm without introducing a large number of branching rules.

1 Introduction

In the line of research on worst-case analysis of exact algorithms for NP-hard problems, the *maximum independent set* problem (MIS) is one of the most important problems. It asks us to find a maximum set of vertices in a graph such that no pair of vertices in the set are adjacent to each other. The method of trivially checking all possible vertex subsets results in an $O^*(2^n)$-time algorithm. In the last half a century, great progresses have been made on exact exponential algorithms and their worst-case analysis for MIS. In 1977, Tarjan and Trojanowski [12] published the first nontrivial $O^*(2^{n/3})$-time algorithm. The bound of the running time to exactly solve the problem has been further improved for many times [8,11,4,9,1]. Currently, the fastest algorithms are Robson's $O^*(1.2109^n)$-time exponential-space algorithm [11] and Bourgeois *et al.*'s $O^*(1.2114^n)$-time polynomial-space algorithm [1].

Most polynomial-space algorithms for MIS use the following simple idea to search a solution: branch on a vertex of maximum degree by either excluding it from the solution set or including it to the solution set. In the first branch we will delete the vertex from the graph and in the second branch we will delete the vertex together with all its neighbors from the graph. When the vertex to be branched on is of degree at least 8, the simple branch is almost good enough

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 72–83, 2013.

to get the running time bound of all published polynomial-space algorithms for MIS. Then MIS in graphs with degree bounded by $i \in \{3, 4, 5, 6, 7\}$ may be the bottleneck cases of MIS. For most cases the running time bound for MIS-i (the maximum independent set problem in graphs with maximum degree i) is one of the bottlenecks to improve the running time bound for MIS-$(i + 1)$, especially for small i. This holds in many algorithms for MIS [1,2,4,9,17]. We look at the most recent two algorithms for MIS in general graphs. Kneis *et al.* [9] used a fast algorithm for MIS-3 by Razgon [10] and used a computer-aided method to check a huge number of cases for MIS-4, and then these two special cases will not be the bottleneck cases in their algorithm for MIS in general graphs. In Bourgeois *et al.*'s paper [1], more than half is discussing algorithms for MIS-3 and MIS-4, and based on improved running time bounds for MIS-3 and MIS-4 they can improve the running time bounds for MIS-5, MIS-6 and then MIS in general graphs. We can see that MIS in low-degree graphs are important. In the literature, we can find a long list of contributions to fast exact algorithms for MIS in low-degree graphs [16,15,10,14,7,5,17]. Currently, MIS-3 can be solved in $O^*(1.0836^n)$ time [16], MIS-4 can be solved in $O^*(1.1376^n)$ time [17], MIS-5 can be solved in $O^*(1.1895^n)$ time and MIS-6 can be solved in $O^*(1.2050^n)$ time [1], where all these use polynomial space. In this paper, we will design an $O^*(1.1737^n)$-time polynomial-space algorithm for MIS-5, improving all previous running time bounds for this problem.

To avoid some bad branches in the algorithm, we may need to reduce some special local structures of the graph. First, we apply our *reduction rules* to find a part of the solution when the graph has certain structures. Second, we design effective divide-and-conquer algorithms based on small cuts of the graph. By reducing the local structures in the above two steps, we can apply our *branching rules* on the graph to search a solution. In our algorithm, the divide-and-conquer methods are newly introduced and they can effectively reduce some bottleneck cases, and we design effective branching rules based on careful check on the structures of the graph and analysis of their properties. These are crucial techniques used in the paper to get the significant improvement on this problem. Due to the limited space, some proofs of lemmas are not included in the extended abstract. Readers are referred to [18] for a full version of this paper.

2 Notation System

Let $G = (V, E)$ stand for a simple undirected graph with a set V of vertices and a set E of edges. Let $n = |V|$. We will use n_i to denote the number vertices of degree i in G, and $\alpha(G)$ to denote the size of a maximum independent set of G. The vertex set and edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively. For simplicity, we may denote a singleton set $\{v\}$ by v and the union $X \cup \{v\}$ of a subset X and an element by $X + v$.

For a subset $X \subseteq V$, let $\overline{X}$ denote the complement set $V \backslash X$, $N(X)$ denote the set of all vertices in $\overline{X}$ that are adjacent to a vertex in X, and $N[X] = X \cup N(X)$. Let $\delta(v) = |N(v)|$ denote the degree of a vertex v. For a subset $X \subseteq V$, $\delta(X)$

denote the sum of degree of vertices in X and $\delta_{\geq 3}(X)$ denote the sum of degree of vertices of degree ≥ 3 in X. We use $N_2(v)$ to denote the set of vertices with distance exactly 2 from v, and let $N_2[v] = N_2(v) \cup N[v]$. Let $G - X$ be the graph obtained from G by removing the vertices in X together with any edges incident to a vertex in X, $G[X] = G - (V \setminus X)$ be the graph induced from G by the vertices in X, and G/X denote the graph obtained from G by contracting X into a single vertex (removing self-loops and parallel edges). For a vertex v, let f_v denote the number of edges between $N[v]$ and $N_2(v)$. In a graph with maximum degree 5, we define the *gain* g_v of v to be

$$g_v = \sum_{t \in N_2(v)} (5 - \delta(t)) + (f_v - |N_2(v)|),$$

where the second term means the number of times two edges leaving $N[v]$ meet at a same vertex in $N_2(v)$. We denote $(f_v, g_v) \geq (a, b)$ when $f_v \geq a$ and $g_v \geq b$ hold.

A partition (V_1, Z, V_2) of the vertex set $V(G)$ of a graph G is called a *separation* if $V(G)$ is a disjoint union of nonempty subsets V_1, Z and V_2 and there is no edge between V_1 and V_2, where Z is called a *vertex cut*. In this paper, a vertex cut is always assumed to be a minimal vertex cut, i.e., no proper subset of a vertex cut is still a vertex cut. The *line graph* of a graph G is the graph whose vertices correspond to the edges of G, and two vertices are adjacent iff the corresponding two edges share a same endpoint in G. Throughout the paper we use a modified O notation that suppresses all polynomially bounded factors. For two functions f and g, we write $f(n) = O^*(g(n))$ if $f(n) = g(n)\text{poly}(n)$ holds for a polynomial $\text{poly}(n)$ in n.

3 Reduction Rules

First of all, we introduce the reduction rules, which can be applied in polynomial time and reduce the graph by finding a part of the solution. There are many reduction rules for MIS and the related *vertex cover* problem [2], from the simplest ones to deal with degree-1 and degree-2 vertices to the somewhat complicated unconfined vertices and crown reductions. They can be found in almost all exact algorithms and most of approximation and heuristic algorithms for MIS. We introduce some reduction rules that will be used our algorithm.

Reduction by Eliminating Easy Instances

For a disconnected graph G with a component H, we see that $\alpha(G) = \alpha(H) + \alpha(G - V(H))$, and solve instances H and $G - V(H)$ independently. We will solve two kinds of components H directly:
(1) H has at most $\rho = 28$ vertices; and
(2) H is the line graph of a bipartite graph H' between the set of degree-3 vertices and the set of degree-4 vertices, which are call a $(3, 4)$-*bipartite graph*.

Case (1) can be solved in constant time since the size of the graph is constant. Case (2) is based on the following observation: if graph G is the line graph of a

graph G', then we obtain a maximum independent set of G directly by finding a maximum matching M in G' and taking the corresponding vertex set V_M in G as a solution. There are several methods to check whether a graph is a line graph or not [13]. To identify a line graph $L(G')$ of a $(3,4)$-bipartite graph G', we need to check if $L(H)$ is a union of 3-cliques and 4-cliques such that each vertex is a common vertex of a 3-clique and a 4-clique.

Reduction by Removing Unconfined Vertices

A vertex v in an instance G is called *removable* if $\alpha(G) = \alpha(G - v)$. A sufficient condition for a vertex to be removable has been studied in [16]. In this paper, we only use a simple case of the condition. For a vertex v and its neighbor $u \in N(v)$, a vertex $s \in V \setminus V[v]$ adjacent to u is called an *out-neighbor* of u. A neighbor $u \in N(v)$ is called an *extending child* of v if u has exactly one out-neighbor $s_u \in V \setminus N[v]$, where s_u is also called an *extending grandchild* of v. Note that an extending grandchild s_u of v may be adjacent to some other neighbor $u' \in N(v) \setminus \{u\}$ of v. Let $N^*(v)$ denote the set of all extending children $u \in N(v)$ of v, and I_v be the set of all extending grandchildren s_u $(u \in N^*(v))$ of v together with v itself. We call v *unconfined* if there is a neighbor $u \in N(v)$ which has no out-neighbor or $I_v \setminus \{v\}$ is not an independent set (i.e., some two vertices in $I_v \cap N_2(v)$ are adjacent). It is known in [16] that any unconfined vertex is removable.

Lemma 1. [16] *For an unconfined vertex v in graph G, it holds that*

$$\alpha(G) = \alpha(G - v).$$

A vertex u *dominates* another vertex v if $N[u] \subseteq N[v]$, where v is called *dominated*. We see that dominated vertices are unconfined vertices.

Reduction by Folding Twins

The set $\{v_1, v_2\}$ of two nonadjacent degree-3 vertices is called a *twin* if $N(v_1) = N(v_2)$.

Lemma 2. [16] *For a twin $A = \{v_1, v_2\}$, we have that*

$$\alpha(G) = \alpha(G^\star) + 2,$$

where $G^\star = G/N[A]$ if $N(A)$ is an independent set and $G^\star = G - N[A]$ otherwise.

Folding a twin $A = \{v_1, v_2\}$ is to remove or contract $N[A]$ in the above way.

Reduction by Folding Short Funnels

A degree-3 vertex v together with its neighbors $N(v) = \{a, b, c\}$ is called a *funnel* if $N[v] \setminus \{a\}$ induces a triangle for some $a \in N(v)$, and the funnel is denoted by a-v-$\{b, c\}$. Note that v dominates any vertex in $N(a) \cap N(v)$ if $N(a) \cap N(v)$ is not empty. When we assume that there are no dominated vertices anymore, then $N(a) \cap N(v) = \emptyset$.

Folding a funnel a-v-$\{b, c\}$ means that we add an edge between every non-adjacent pair (x, y) of vertices $x \in N(a) \setminus \{v\}$ and $y \in \{b, c\}$ and then remove vertices a and v.

Let $G^\dagger$ denote the graph after folding a funnel a-v-$\{b, c\}$ in G. Then we have the following lemma.

Lemma 3. [16] *For any funnel a-v-$\{b, c\}$ in graph G, it holds that*

$$\alpha(G) = 1 + \alpha(G^\dagger).$$

We call a funnel a-v-$\{b, c\}$ in a graph with minimum degree 3 a *short funnel* if $\delta(a) = 3$ (resp., $\delta(a) = 4$) and between $N(a) \setminus \{v\}$ and $\{b, c\}$ there is at least one edge (resp., there are at least two edges meeting at the same vertex b or c). In our algorithm, we will reduce short funnels only and leave some other funnels.

Definition 1. *A graph is called a* reduced graph *if none of the above reduction operations is applicable.*

The algorithm in Figure 1 is a collection of all above reduction operations. When the graph is not a reduced graph, we can use the the algorithm in Figure 1 to reduce it and find a part of the solution.

Input: A graph $G = (V, E)$ and the size s of the current partial solution (initially $s = 0$).
Output: A reduced graph $G' = (V', E')$ and the size s' of a partial solution S' with $N[S'] \cap V' = \emptyset$ in G.

1. **If** {Graph G has a component H that is a graph with at most $\rho = 28$ vertices or the line graph of a $(3, 4)$-bipartite graph}, **return** $(G', s') := \mathrm{RG}(G - V(H), s + \alpha(H))$.
2. **Elseif** {There is an unconfined vertex v}, **return** $(G', s') := \mathrm{RG}(G - v, s)$.
3. **Elseif** {There is a twin $A = \{u, v\}$}, **return** $(G', s') := \mathrm{RG}(G^\star, s + 1)$ for $G^\star = G - N[A]$ if $N(v)$ is an independent set, and $G^\star = G/N[A]$ otherwise.
4. **Elseif** {There is a short funnel}, **return** $(G', s') := \mathrm{RG}(G^\dagger, s + 1)$.
5. **Else return** $(G', s') := (G, s)$.

Fig. 1. The Algorithm $\mathrm{RG}(G, s)$

4 Properties of Vertex-Cuts with Size at Most 2

For a disconnected graph G with a component H, we can solve instances H and $G - V(H)$ independently. Here we observe a similar property on graphs with vertex-connectivity 1 and 2.

Let v be a vertex cut in a graph G, which gives a separation $(V_1, \{v\}, V_2)$. Let $G_i = G[V_i]$, $i = 1, 2$, and $V_1^v = V_1 \setminus N(v)$. The induced graph $G[V_1^v]$ is denoted by G_1^v.

The following theorems provide a divide-and-conquer method for us to find a maximum independent set in G.

Theorem 1. *For subgraphs G_1 and G_1^v defined on a separation $(V_1, \{v\}, V_2)$ in a graph G, it holds*

$$\alpha(G) = \alpha(G_1) + \alpha(G^\star),$$

where $G^\star = G - V_1$ if $\alpha(G_1) = \alpha(G_1^v)$, and $G^\star = G_2$ otherwise. A maximum independent set in a graph G can be constructed from any maximum independent sets to G_1, G_1^v and $G^\star$.

For a separation $(V_1, \{u, v\}, V_2)$ of a graph G, let $G_i = G[V_i]$ $(i = 1, 2)$, $V_1^v = V_1 \backslash N(v)$, $V_1^u = V_1 \backslash N(u)$ and $V_1^{uv} = V_i \backslash N(\{u, v\})$, $i \in \{1, 2\}$. The induced graphs $G[V_1^v]$, $G[V_1^u]$ and $G[V_1^{uv}]$ are simply denoted by G_1^v, G_1^u and G_1^{uv} respectively. Let $\widetilde{G_2}$ denote the graph obtained from $G[V_2 \cup \{u, v\}]$ by adding an edge uv if v and u are not adjacent.

Theorem 2. *For subgraphs G_1, G_1^v, G_1^u and G_1^{uv} defined on a separation $(V_1, \{u, v\}, V_2)$ in a graph G, it holds*

$$\alpha(G) = \alpha(G_1) + \alpha(G^\star),$$

where

$$G^\star = \begin{cases} G[V_2 \cup \{u,v\}] & \text{if } \alpha(G_1^{uv}) = \alpha(G_1), \\ \widetilde{G_2} & \text{if } \alpha(G_1^{uv}) < \alpha(G_1^u) = \alpha(G_1^v) = \alpha(G_1), \\ G[V_2 + v] & \text{if } \alpha(G_1^u) < \alpha(G_1^v) = \alpha(G_1), \\ G[V_2 + u] & \text{if } \alpha(G_1^v) < \alpha(G_1^u) = \alpha(G_1), \\ G/(V_1 \cup \{u, v\}) & \text{if } \alpha(G_1^{uv}) + 1 = \alpha(G_1) \text{ and } \alpha(G_1^v) < \alpha(G_1), \\ G_2 & \text{otherwise } (\alpha(G_1^{uv}) + 2 \leq \alpha(G_1) \text{ and } \alpha(G_1^v) < \alpha(G_1)). \end{cases}$$

A maximum independent set in a graph G can be constructed from any maximum independent sets to G_1, G_1^v, G_1^u, G_1^{uv} and $G^\star$.

Note that the above divide-and-conquer method can be used to deal with degree-1 and degree-2 vertices in the graph.

5 Branching Rules

We introduce our branching rules, which will only be applied on a graph that is reduced and connected component of it is triconnected.

Branching on a Vertex

A simple branching rule is to branch on a single vertex v by considering two cases: (i) there is a maximum independent set of G which does not contain v; (ii) every maximum independent set of G contains v. In (ii), it is shown that I_v is always contained in any maximum independent set of G [16]. Recall that I_v is the set of all extending grandchildren of v together with v itself.

Branching on a vertex v means creating two subinstances by excluding v from the independent set or including I_v to the independent set. In the first branch we will delete v from the instance whereas in the second branch we will delete $N[I_v]$ from the instance. Selecting vertices to branch on is important for efficiency of our algorithms.

A vertex cut Z of size $|Z| = 3$ is a *good* vertex cut if there is a separation (X_1, Z, X_2) such that

$$|X_1| \leq 24,\ \delta(X_1) \geq 17 \text{ and } X_1 \text{ induces a connected subgraph,}$$

and X_1 is maximal under the above two conditions. A pair of nonadjacent vertices u and v is called a *good* pair if one of u and v is a degree-5 vertex, and u and v share at least three common neighbors. A funnel a-v-$\{b, c\}$ is called a *good* funnel if vertex a has a neighbor u such that (v, u) is a good pair. The vertices on which we branch will be chosen as follows:

(i) the vertex is a vertex in a good vertex cut;
(ii) the vertex is a in a good funnel a-v-$\{b, c\}$; or
(iii) the vertex is a vertex of maximum degree $d \geq 5$.

When we branch on the vertex a of a funnel a-v-$\{b, c\}$ in (iii), we generate instance by excluding a or by including I_a. In the first branch, after removing a, vertex v becomes dominated, and we can include it in a solution. Therefore we get the following branching rule [16].

Branching on a funnel a-v-$\{b, c\}$ in a reduced instance G by either including v or including I_a in the independent set. Hence we generate the two subinstances by removing either $N[v]$ or $N[I_a]$ from G.

Branching on a Complete Bipartite Subgraph

Lemma 4. *Let A and B be two disjoint vertex subsets in a graph G such that every two vertices $a \in A$ and $b \in B$ are adjacent. Then either $S \cap A = \emptyset$ or $S \cap B = \emptyset$ holds for any independent set S in G.*

Proof. If $S \cap A = \emptyset$ then we are done. If S contains a vertex $a \in A$, then $S \cap N(a)$ $(\supseteq S \cap B)$ is empty. ∎

For a good pair $\{u, v\}$, we have a bipartite graph between $A = \{u, v\}$ and $B = N(u) \cap N(v)$.

Branching on a good pair $\{u, v\}$ means branching by either excluding $\{u, v\}$ from the independent set or excluding $N(u) \cap N(v)$ from the independent set.

6 The Algorithm and Results

6.1 Framework for Analysis

We apply the Measure and Conquer method [4] to analyze our algorithm. In this method, we introduce a weight to each vertex in the graph according to

the degree of the vertex, $w : \mathbb{Z}_+ \to \mathbb{R}_+$ (where $\mathbb{Z}_+$ and $\mathbb{R}_+$ denote the sets of nonnegative integers and nonnegative reals, respectively): we denote by w_i the weight $w(v)$ of a vertex v of each degree $i \geq 0$ and let $\Delta w_i = w_i - w_{i-1}$ for $i \geq 1$. We will set vertex weight such that: $\Delta w_5 \leq \Delta w_i$ for $i \geq 3$, vertices of higher degree receive a larger weight (i.e., $\Delta w_i \geq 0$), the problem can be solved in polynomial time when $\mu(G) \leq 0$, and the weight of each vertex in the initial graph G is not greater than 1 (i.e., $\mu(G) \leq n$). Then we adopt $\mu(G) = \sum_i w_i n_i$ as the measure of the graph G. To analyze our algorithm, we will construct a recurrence related to the measure $\mu = \mu(G)$ for each branch in our algorithm and analyze a bound for the worst ones.

For each branch operation, we will generate two subinstances G_1 and G_2 by deleting some vertices from the graph. After deleting some vertices, we can reduce the measure from two parts: the weight of the vertices being deleted and partial weight of the vertices adjacent to the deleted vertices since their degree will decrease. Let $t_{(i)}$ be a lower bound on the decrease of the measure in the subinstance (i.e., $\mu(G) - \mu(G_i) \geq t_{(i)}$). Then we get the recurrence $C(\mu) \leq C(\mu - t_{(1)}) + C(\mu - t_{(2)})$. The unique positive real root of the function $f(x) = 1 - x^{-t_{(1)}} - x^{-t_{(2)}}$ is called the *branching factor* of the above recurrence. Let τ be the maximum branching factor among all branching factors in the search tree. Then $C(\mu) = \tau^w$. Readers are referred to [6] for more details about the analysis.

The most important and complicated case in our algorithm is to branch on a vertex v of maximum degree. Let $\Delta_{out}(v)$ and $\Delta_{in}(v)$ to denote the decrease of the measure of μ in the branches of excluding v and including I_v, respectively. We get recurrence $C(\mu) = C(\mu - \Delta_{out}(v)) + C(\mu - \Delta_{in}(v))$. We give more details about lower bounds on $\Delta_{out}(v)$ and $\Delta_{in}(v)$. Let k_i denote the number of degree-i neighbors of v. Then $d = \sum_{i=3}^{d} k_i$. For the first branch, we get

$$\Delta_{out}(v) = w_d + \sum_{i=3}^{d} k_i \Delta w_i.$$

In the second branch, we will delete $N[I_v]$ from the graph. Let $\Delta(\overline{N[v]})$ denote the decrease of weight of vertices in $V(G) \setminus N[v]$ by removing $N[I_v]$ from G together with possibly weight decrease attained by reduction operations applied to $G - N[I_v]$. Then we have

$$\Delta_{in}(v) \geq w_d + \sum_{i=3}^{d} k_i w_i + \Delta(\overline{N[v]}).$$

We can branch on a vertex v of degree d with recurrence

$$\begin{aligned} C(\mu) &= C(\mu - \Delta_{out}(v)) + C(\mu - \Delta_{in}(v)) \\ &\leq C(\mu - (w_d + \textstyle\sum_{i=3}^{d} k_i \Delta w_i) + C(\mu - (w_d + \sum_{i=3}^{d} k_i w_i + \Delta(\overline{N[v]}))). \end{aligned} \quad (1)$$

In our algorithm, we carefully select a vertex of maximum degree to branch on so that the worst recurrence (1) is as good as possible. To do so, we need to analyze lower bounds on $\Delta(\overline{N[v]})$ when the maximum degree of the graph is 5. If no vertex in $N_2(v)$ is adjacent to two vertices in $N_1(v)$, then $\Delta(\overline{N[v]}) \geq f_v \Delta w_5$, since we assume that $\Delta w_5 \leq \Delta w_i$ for $i \geq 3$. Otherwise, we have the following lower bound based on our weight setting (the proof can be found in the full version of this paper):

$$\Delta(\overline{N[v]}) \geq f_v \Delta w_5 + g_v(\Delta w_4 - \Delta w_5). \tag{2}$$

When $N^*(v) \neq \emptyset$ (v has some extending children), the above bound may not be good enough since f_v may be small. For this case, $N[I_v]$ will be removed from the graph in the second branch and $\Delta(\overline{N[v]})$ can reach a desired bound.

6.2 The Algorithm

Our algorithm is simple in the sense that it consists of branching on three kinds of vertices and branching on good pairs except for how to select vertices of maximum degree 5 to branch on. A reduced degree-5 graph is a *proper graph* if it has neither good vertex cuts nor good pairs and each connected component of it has vertex connectivity at least 3. In fact, branching on a vertex v of maximum degree 5 in a proper graph will be bottlenecks in the analysis for the running time bound of our algorithms. We here identify degree-5 vertices v in proper graphs branching on which would efficiently reduce the current instance in terms of the degrees of neighbors of v. These vertices are called *optimal* vertices.

For a degree-5 vertex v in a proper graph, let $k(v) = (k_3, k_4, k_5)$, where k_i is the number of degree-i neighbors of v ($i = 3, 4, 5$).

The vertex v is called *effective* if one of the following (a)-(f) holds:

(a) $(f_v, g_v) \geq (14, 0)$, $(12, 3)$ or $(10, 5)$, for $(k_4, k_5) = (0, 5)$;
(b) $(f_v, g_v) \geq (13, 0)$ or $(11, 2)$, for $(k_4, k_5) = (1, 4)$;
(c) $(f_v, g_v) \geq (12, 0)$ or $(10, 2)$, for $(k_4, k_5) = (2, 3)$;
(d) $(f_v, g_v) \geq (11, 0)$, for $(k_4, k_5) = (3, 2)$;
(e) $(f_v, g_v) \geq (12, 0)$ or $(10, 1)$, for $(k_4, k_5) = (4, 1)$; and
(f) $(f_v, g_v) \geq (10, 0)$, for $(k_4, k_5) = (5, 0)$.

Lemma 5. *Let G be a proper graph with at least one degree-5 vertex. Assume that $N^*(u) = \emptyset$ for all degree-5 vertices in G and that no degree-5 vertex is adjacent to a degree-3 vertex. Then there exists an effective vertex in G.*

A degree-5 vertex v in a proper graph is called *optimal* if either (1) $k_3 \geq 1$ or (2) there is no degree-5 vertex adjacent to a degree-3 vertex, and v is effective or it holds $N^*(v) \neq \emptyset$. Our algorithm for MIS-5 is described in Figure 2.

Input: A graph G.
Output: The size of a maximum independent set in G.

1. **If** {Graph G has a vertex cut v with a separation $(V_1, \{v\}, V_2)$ such that $\delta_{\geq 3}(V_1) \leq \delta_{\geq 3}(V_2)$}, **return** MIS5($G[V_1]$) + MIS5($G^\star$).
2. **Elseif** {Graph G has a vertex cut $\{u, v\}$ with a separation $(V_1, \{u, v\}, V_2)$ such that $\delta_{\geq 3}(V_1) \leq \delta_{\geq 3}(V_2)$}, **return** MIS5($G[V_1]$) + MIS5($G^\star$).
3. **Else** Let $(G, s) := \mathrm{RG}(G, 0)$.
4. **If** {G has a vertex of degree ≥ 6}, pick up a vertex v of maximum degree, and **return** $s + \max\{\mathrm{MIS5}(G - v), |I_v| + \mathrm{MIS5}(G - N[I_v])\}$.
5. **Elseif** {G has a good vertex cut}, pick up a vertex v in a good vertex cut, and **return** $s + \max\{\mathrm{MIS5}(G - v), |I_v| + \mathrm{MIS5}(G - N[I_v])\}$.
6. **Elseif** {G has a good funnel a-v-$\{b, c\}$, **return** $s + \max\{1 + \mathrm{MIS5}(G - N[I_a]), 1 + \mathrm{MIS5}(G - N[v])\}$.
7. **Elseif**{G has a good pair (u, v)}, **return** $s + \max\{\mathrm{MIS5}(G - \{u, v\}), \mathrm{MIS5}(G - N(u) \cap N(v))\}$.
8. **Elseif** {G has a degree-5 vertex}, pick up an optimal degree-5 vertex v, and **return** $s + \max\{\mathrm{MIS5}(G - v), |I_v| + \mathrm{MIS5}(G - N[I_v])\}$.
9. **Else** {G is a degree-4 graph}, use the algorithm for MIS-4 in [17] to solve the instance G and **return** $s + \alpha(G)$.

Note: With a few modifications, the algorithm can deliver a maximum independent set.

Fig. 2. Algorithm MIS5(G)

6.3 The Result

In our algorithm, we set vertex weight as follows

$$w_i = \begin{cases} 0 & \text{for } i = 0, 1 \text{ and } 2 \\ 0.5091 & \text{for } i = 3 \\ 0.8243 & \text{for } i = 4 \\ 1 & \text{for } i = 5 \\ 1.5091 & \text{for } i = 6 \\ 1.7482 & \text{for } i = 7 \\ 1.9722 & \text{for } i = 8 \\ w_8 + (i-8)(w_5 - w_4) & \text{for } i \geq 9. \end{cases} \tag{3}$$

Lemma 6. *With the vertex weight setting (3), each recurrence generated by the algorithm in Figure 2 has an amortized branching factor not greater than* 1.1737.

The main idea of the proof is given in the next section. From Lemma 6 we know that the size of the search tree generated by our algorithm is not greater than 1.1737^μ, where μ is not greater than the number n of vertices in the initial graph since it has maximum degree 5.

Theorem 3. *A maximum independent set in a degree-*5 *graph of* n *vertices can be found in* $O^*(1.1737^n)$ *time.*

7 Framework for the Proof of Lemma 6

To prove Lemma 6, we need to analyze each step of our algorithm and find out the worst recurrences generated by them under the weight setting (3). A full proof of the analytical lemma can be found in the full version of this paper [18]. Here we give some main lemmas used to prove it.

Under the vertex weight setting (3), we can verify that

Lemma 7. *$w_i + w_j \geq w_{i+j-2}$ holds for all $i, j \geq 1$, and $w_i + w_j \geq w_{i+j-2} + \Delta w_5$ if $i + j - 2 \leq 5$.*

Based on Lemma 7, we can show that the measure never increases in $\mathrm{RG}(G, s)$.

Lemma 8. *The measure μ of a graph G never increases in $RG(G, s)$. Moreover μ decreases by at least Δw_5 after any step in $RG(G, s)$ if the maximum degree decreases by at least one after this step.*

Next, we will analyze the recurrences created in each step of the algorithm $\mathrm{MIS5}(G)$.

Lemma 9. *The divide-and-conquer process in Steps 1 and 2 of* $\mathrm{MIS5}(G)$ *will not generate a recurrence with branching factor greater than 1.1737.*

Lemma 10. *None of branching on a vertex of maximum degree ≥ 6 in Step 4, branching on a vertex in a good vertex cut in Step 5, branching on a good funnel in Step 6, and branching on a good pair in Step 7 of* $\mathrm{MIS5}(G)$ *will generate a recurrence with branching factor greater than 1.1737.*

After Step 7 of $\mathrm{MIS5}(G)$, the graph is a proper graph if it still has some vertices of degree 5. We can prove that

Lemma 11. *Let v be an optimal degree-5 vertex in a proper graph G. Then it holds*

$$\Delta(\overline{N[v]}) \geq \lambda(k_3, k_4, k_5),$$

where

$$\lambda(k_3, k_4, k_5) = \begin{cases} 14\Delta w_5 & \text{if } (k_3, k_4, k_5) = (0, 0, 5) \\ 9\Delta w_5 + 2\Delta w_4 & \text{if } (k_3, k_4, k_5) = (0, 1, 4) \\ 8\Delta w_5 + 2\Delta w_4 & \text{if } (k_3, k_4, k_5) = (0, 2, 3) \\ 9\Delta w_5 + \Delta w_4 & \text{if } (k_3, k_4, k_5) = (0, 4, 1) \\ 11\Delta w_5 & \text{if } (k_3, k_4, k_5) = (0, 3, 2) \\ 10\Delta w_5 & \text{if } (k_3, k_4, k_5) = (0, 5, 0) \\ 2w_3 + 6\Delta w_5 & \text{if } k_3 \geq 2 \text{ and } N(v) \setminus N^*(v) \text{ contains no degree-3 neighbor of } v \\ 10\Delta w_5 & \text{if } k_3 = 1 \text{ or } N(v) \setminus N^*(v) \text{ contains a degree-3 neighbor of } v. \end{cases}$$

Based on a refined analysis of (1) and Lemma 11, we can prove that

Lemma 12. *Branching on an optimal degree-5 vertex in Step 8 of* MIS5(G) *will not generate a recurrence with branching factor greater than 1.1737 (with amortization).*

When the graph becomes a graph of degree ≤ 4, the algorithm will use the $O^*(1.1376^n)$-time algorithm for MIS-4 in [17] to solve it in Step 9. We can verify that under (3), the step is also good enough to get the bound of 1.1737.

References

1. Bourgeois, N., Escoffier, B., Paschos, V.T., van Rooij, J.M.M.: Fast algorithms for max independent set. Algorithmica 62(1-2), 382–415 (2012)
2. Chen, J., Kanj, I.A., Xia, G.: Improved upper bounds for vertex cover. Theoretical Computer Science 411(40-42), 3736–3756 (2010)
3. Eppstein, D.: Quasiconvex analysis of backtracking algorithms. In: SODA, pp. 781–790. ACM Press (2004)
4. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM 56(5), 1–32 (2009)
5. Fomin, F.V., Høie, K.: Pathwidth of cubic graphs and exact algorithms. Inf. Process. Lett. 97(5), 191–196 (2006)
6. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Springer (2010)
7. Fürer, M.: A faster algorithm for finding maximum independent sets in sparse graphs. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 491–501. Springer, Heidelberg (2006)
8. Jian, T.: An $O(2^{0.304n})$ algorithm for solving maximum independent set problem. IEEE Transactions on Computers 35(9), 847–851 (1986)
9. Kneis, J., Langer, A., Rossmanith, P.: A fine-grained analysis of a simple independent set algorithm. In: Kannan, R., Kumar, K.N. (eds.) FSTTCS 2009, Dagstuhl, Germany. LIPIcs, vol. 4, pp. 287–298 (2009)
10. Razgon, I.: Faster computation of maximum independent set and parameterized vertex cover for graphs with maximum degree 3. J. of Discrete Algorithms 7(2), 191–212 (2009)
11. Robson, J.: Algorithms for maximum independent sets. J. of Algorithms 7(3), 425–440 (1986)
12. Tarjan, R., Trojanowski, A.: Finding a maximum independent set. SIAM J. on Computing 6(3), 537–546 (1977)
13. West, D.: Introduction to Graph Theory. Prentice Hall (1996)
14. Xiao, M., Chen, J.E., Han, X.L.: Improvement on vertex cover and independent set problems for low-degree graphs. Chinese J. of Computers 28(2), 153–160 (2005)
15. Xiao, M.: A simple and fast algorithm for maximum independent set in 3-degree graphs. In: Rahman, M. S., Fujita, S. (eds.) WALCOM 2010. LNCS, vol. 5942, pp. 281–292. Springer, Heidelberg (2010)
16. Xiao, M., Nagamochi, H.: Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. Theoretical Computer Science 469, 92–104 (2013)
17. Xiao, M., Nagamochi, H.: A refined algorithm for maximum independent set in degree-4 graphs. Technical report 2013-002, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University (2013), `http://www.amp.i.kyoto-u.ac.jp/tecrep/abst/2013/2013-002.html`
18. Xiao, M., Nagamochi, H.: An Exact Algorithm for Maximum Independent Set in Degree-5 Graphs. Technical report 2013-003, Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University (2013), `http://www.amp.i.kyoto-u.ac.jp/tecrep/abst/2013/2013-003.html`

FWLS: A Local Search for Graph Coloring

Wei Wu[1], Chuan Luo[1], and Kaile Su[2,3]

[1] Key Laboratory of High Confidence Software Technologies, Ministry of Education, Institute of Software, School of Electronic Engineering and Computer Science, Peking University, Beijing, China
wuwei_eecs@pku.edu.cn, chuanluosaber@gmail.com
[2] College of Mathematics Physics and Information Engineering, Zhejiang Normal University, Jinhua, China
[3] Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
kailepku@gmail.com

Abstract. Local search (LS) is a widely used, general approach for solving hard combinatorial search problems, such as the graph coloring problem (GCP). One main advantage of this method is that effective heuristics for a problem may lead to improvements in solving other problems. Recently, it has been shown that an initial LS algorithm for the Boolean satisfiability problem (SAT) called WalkSAT is extremely effective for random SAT instances. Thus, it is interesting to apply the heuristics in WalkSAT to GCP. This paper proposes a random walk based heuristic, which is inspired by WalkSAT but differs in the tie-breaking mechanism. This new heuristic leads to a new LS algorithm for GCP namely FWLS. The experiments on the DIMACS benchmark show that FWLS finds optimal (or best known) solutions for most instances. Also, when compared to other GCP algorithms, including a greedy one, an LS one and a hybrid one, FWLS exhibits very competitive or better performance.

1 Introduction

The *graph coloring problem* (GCP) is a prominent NP-hard problem [1]. Given an undirected graph $G = (V, E)$ with a set V of vertices and a set E of edges, the task is to color each vertex of G in such a manner that there exist no two adjacent vertices with the same color. GCP is to minimize the number of colors used, and the minimum number of colors is called *chromatic number* χ of G, $\chi(G)$.

GCP has many practical applications such as frequency assignments, register allocations, timetabling, satellite range scheduling as well as manufacturing [2–6]. Due to its significance in applications, many methods have been proposed to solve GCP. The elaborated exact methods can find optimal colorings, but they are all time-consuming and not applicable. Thus, heuristic and metaheuristic methods have been developed to find optimal solutions on larger instances. Heuristic and metaheuristic methods can be divided into three categories: greedy constructive algorithms, local search (LS) algorithms and hybrid algorithms which combine

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 84–93, 2013.

an LS method with an evolutionary algorithm. A greedy constructive algorithm colors vertices successively following some rule used for choosing the next vertex to be colored. Famous greedy algorithms include DSATUR [9] and RLF [29] proposed by Brélaz and Leighton respectively in 1979. An LS algorithm starts from a solution (proper or improper) of the given problem, and then iteratively moves from the current solution to a neighboring one in the subsequent search steps to find better solutions measured by an appropriate object function. The neighborhood $N(S)$ of a solution S is a set of solutions which are largely close to S and can be easily obtained. An evolutionary algorithm is mainly built on population-based algorithms such as genetic algorithms and ant algorithms, which simulate biologic characteristics added to some strategies to diversify solutions such that algorithms are able to escape from local optima. So far in the field of GCP, LS algorithms and evolutionary algorithms are best studied [7, 8, 11, 12, 17–24, 28]. Additionally, one significant strategy called *tabu*, in which some neighbors would be forbidden to visit in the next several moves, is also widely used in GCP since Hertz and de Werra proposed Tabucol in 1987 [7], giving rise to many improved algorithms [8, 21–24].

The *satisfiability problem* (SAT), another well-known NP-hard problem, is to determine for a given propositional formula in conjunctive normal form (CNF) with variables $\{x_1, x_2, ...x_N\}$ whether there is an assignment for the variables such that all clauses are satisfied [1]. A well-known LS algorithm for SAT is the WalkSAT algorithm, which only focuses on the variables to be flipped from the unsatisfied clauses contributing to narrowing the flip, and then makes greedy or random flips with a fixed probability, breaking ties at random [26]. S.Bounkong et al. made use of a hybrid algorithm combining WalkSAT with GSAT to solve *graph-coloring-like problem*, i.e., maximizing the number of colors at one edge distance of any vertex in the random graphs [27]. In this paper, we propose Focused Walk based Local Search (FWLS), which is inspired by WalkSAT. FWLS chooses an uncovered vertex greedily or randomly, breaking ties by heuristic information *age* instead of random, i.e., selecting the least recent uncolored vertex in several equivalent ones [25]. Experiments on the graphs of DIMACS challenge benchmark show that FWLS achieves competitive results compared with three algorithms including a greedy one, an LS one and a hybrid one. From the experimental results, our algorithm is able to find the known optimal solutions or the current optimal ones which are obtained by some other algorithms for most of graphs, and also to gain solutions largely close to the current optimal ones for other graphs.

The paper is organized as follows. In section 2, we give some definitions and notations. Section 3 describes the FWLS algorithm, and the experimental results are reported and compared with some other algorithms in Section 4. Finally, we summarize our work and give future directions in Section 5.

2 Preliminary

A *graph* $G = (V, E)$ consists of a set of vertices $V = \{v_1, v_2, ..., v_n\}$, and a set of edges $E \subseteq V \times V$. Given an integer k, the *k-coloring problem* is to find a

partition of V into $\{C_1, C_2, ..., C_k\}$, called *color classes*, each of which stands for a kind of color such that any two vertices in the same *color class* have no shared edge. Formally, a mapping $F : V(G) \to \{C_1, C_2, ..., C_k\}$ is called an *assignment* F.

If there exists a partition with the size of k which meets the constraints of GCP, i.e., all vertices in G can be mapped to a value of *color class* through mapping F, we call it a proper *k-coloring*. Otherwise, this is an improper coloring containing *conflicts* in which two adjacent vertices x and y belong in the same *color class*, i.e., $\exists x \exists y \left((x, y) \in E \wedge F(x) = F(y)\right)$, and the two vertices are called *conflicting vertices*.

3 Focused Walk Based Local Search for Graph Coloring

GCP is a *combinatorial optimization problem*, but it can be solved from the point view of constraint satisfaction by determining whether there exists a solution for *k-coloring problem* [28]. If a proper *k-coloring* is found, the algorithm can be restarted to solve the $(k-1)$-*coloring problem* until no proper *k-coloring* can be found. Therefore, the paper focuses on the *k-coloring problem*.

3.1 Basic Notation and Definitions

Definition 1. *A partial k-coloring of graph $G = (V, E)$ consists of k disjoint color classes, $C_1, C_2, ..., C_k$, and a set $\Delta = V \setminus \bigcup_{i=1}^{k} C_i$, called conflict set, where the set composed of $C_1, C_2, ..., C_k$ is a proper k-coloring for subgraph induced by vertices from $C_1, C_2, ..., C_k$.*

In the above definition, each vertex $v \in V$ exactly belongs to one set of $C_1, C_2, ..., C_k, \Delta$. If $v \in C_i$, v is labeled *color i*. Also if $v \in \Delta$, v has not been colored temporarily. Thus, a *proper k-coloring* is proper if and only if Δ is $\emptyset$. From the viewpoint of mapping, uncolored vertices are mapped to Δ. According to the definition of proper *k-coloring*, the object function is to minimize the size of Δ, and the optimum is 0.

Besides, we use a tuple $<v, color, conflictNum>$ to indicate the number of conflicts $conflictNum$ between the uncolored vertex v and the color class C_{color} if v is colored with *color*. Meanwhile, each uncolored vertex contains a variable *age* related to its tuple, which records steps that have occurred since the vertex entered Δ recently or its *color* in tuple was changed.

During the search procedure, FWLS always maintains a current *partial k-coloring*. Each step to a neighboring solution is to choose one vertex v from Δ and put it into one *color class* C_i. As this action would cause conflicts, some vertices in C_i may enter Δ which are adjacent to v, ensuring that the *k color classes* are always proper. And the tuple $<v, color, conflictNum>$ greedily records a kind of color with the smallest number of conflicts except the color which the vertex v owned recently. Evidently, FWLS prefers to color a certain vertex which produces the smallest number of *conflict vertices*.

There is a parameter *noise* p in FWLS used for selecting greedily or randomly when no vertices with $conflictNum = 0$ [16, 14, 13, 10]. With the probability p it chooses one vertex with the smallest number of conflicts from Δ greedily; otherwise, it chooses one randomly.

3.2 The FWLS Algorithm

The core idea of FWLS is to find a *partial k-coloring* that can be extended to a proper *k-coloring* through moving vertices from Δ to some *color class* C_i, and eliminating *conflicting vertices* in C_i. At first, FWLS completes the initialization for k *color classes* and Δ by a simple method (lines 3-4). We choose a vertex in an ascending order by the serial number of vertex and put it into a C_i with the smallest possible i such that the C_i is still proper. If no such C_i exists, the vertex is placed into Δ. Then FWLS starts to initialize *age* to 0 in Δ and compute the best value of tuple $<v, color, conflictNum>$ for each vertex, during which FWLS tries to endow all vertices in Δ with each color, calculates $conflictNum$ of vertices in corresponding to color and selects the color with the smallest $conflictNum$.

After initialization, FWLS proceeds to extend the *partial k-coloring* by focusing on Δ. Δ are divided into two categories according to whether there exist vertices with $conflictNum = 0$ in it. If Δ has such vertices, FWLS selects one of them to intensify the current optimum; otherwise, it indicates that FWLS is trapped into a local optimum, and requires diversifying, for example, by perturbations.

This is a crucial step to escape from local optima. When there exist no vertices with $conflictNum = 0$ which decrease Δ obviously, this situation indicates that FWLS gets stuck in local minima. In this case, FWLS introduces a parameter *noise* p which determines how often non-greedy steps are taken into consideration during LS. During the diversification stage, with probability p, FWLS chooses a vertex from Δ at random, and in the remaining case it selects a vertex with the smallest number of $conflictNum$ as before, which makes the increment of the size of Δ as least as possible (lines 9-15).

Note that when selecting a vertex with the minimum $conflictNum$, if there are more than one vertices with the equal minimum $conflictNum$, FWLS breaks ties as follows. A vertex owning larger *age* shows that it has been neglected for long, so FWLS will select it to strengthen diversification. But in WalkSAT, it insists on choosing randomly to break ties as before being trapped into a local optimum.

After coloring a certain vertex, FWLS updates Δ accordingly. Then, it increases *age* by 1 for the existing vertices whose colors in tuples are not still changed and sets them to 0 for the added vertices and the existing vertices owing the modified color in Δ (lines 16-18). When updating the existing vertices, FWLS takes full advantage of tuple information in Δ: if a vertex stores color in its tuple which is not given to the selected vertex, FWLS will compare $conflictNum$ produced by the two colors directly and select the less one; otherwise, FWLS firstly modifies $conflictNum$ in corresponding to this color, and

then traverses the remaining *color classes* and chooses the smallest values from them. Hence, FWLS repeats the steps of selecting and eliminating vertices until it seeks out a proper *k-coloring* or reaches the step limit. If FWLS finds a proper *k-coloring*, it outputs the solution; otherwise it declares $Unknown$.

Based on the above description, we outline the FWLS algorithm as Algorithm 1, and give further comments on the algorithm below.

Algorithm 1. FWLS

```
1  FWLS(G, maxStep)
   Input: Graph G, maxSteps
   Output: A proper k-coloring α of G or Unknown
2  begin
3      generate a proper partial k-coloring α;
4      compute < v, color, conflictNum > for each uncolored vertex in conflict set
       Δ; initialize age as 0 for each uncolored vertex;
5      for step ← 1 to maxStep do
6          if α is a proper k-coloring then return α;
7          if there exist vertices with conflictNum = 0 in Δ then
8              select it, breaking ties by preferring the largest age;
9          else if with the fixed probability p then
10             select a vertex randomly in Δ,
11             eliminate vertices colliding with this added one and put them in Δ;
12         else
13             select a vertex with the largest conflictNum,
14             breaking ties by preferring the largest age,
15             eliminate vertices colliding with this added one and put them in Δ;
16         update Δ, compute the optimal tuples,
17         increase age by 1 for the unchanged tuples in Δ
18         and set age to 0 for the added vertices and changed tuples;
19     return Unknown;
```

FWLS keeps a balance between intensification and diversification. In each step, the vertex to be colored is chosen in accordance with heuristic information from the tuple <$v, color, conflictNum$>; otherwise, one vertex is selected randomly. In addition, FWLS utilizes *old-priority* strategy to break ties when faced with several *equal-conflictNum* vertices, which prefers the least recent uncolored vertex. To obtain more diversification, FWLS also takes a random step when trapping into local optima.

4 Experimental Results

In this section, we present intensive experimental results of FWLS algorithm on the well-known DIMACS coloring benchmark given at the COLOR02 workshop[1].

[1] `http://mat.gsia.cmu.edu/COLOR02/`

The algorithm is implemented in C++ and statically compiled by g++ with the '-O2' option on a machine with Intel Core i7 1.6GHz and 8GB RAM under Linux. Further, we run a benchmark program (dfmax) together with a benchmark instance (r500.5) in the COLOR02 workshop, and our machine reports a user time of 7.32s on r500.5.

We test all graphs from DIMACS, and find that FWLS is able to get good performance of most graphs, including series of "DSJC", "flat", "fpsol", "inithx", "le", "mugg", "mulsol", "myciel", "qg.order", "queen", "wap"and "zeroin", as well as three independent instances such as "will199GPIA", "school1_nsh"and "school1". For all instances, the parameter *noise p* is set to 0.5 for FWLS, and we run FWLS 20 times on each instance, terminating upon either finding a proper *k-coloring* or exceeding a given cutoff time which is set to 3800 seconds.

We compare FWLS with some other heuristic algorithms proposed by Daniel et al. (DS), a greedy algorithm [9], Michael et al. (ES), an LS algorithm [11], and Philippe et al. (AMA), a hybrid algorithm [12]. We list the best solutions of these algorithms, the number of successful runs ('#suc') as well as average time ('avg time') for each instance for FWLS in Table 1. Table 1 also shows the number $|V|$ of vertices, the known *chromatic number* χ, and the value k^* which is the current smallest k found by some algorithm. However, because the authors of these three algorithms did not provide binary codes for their algorithms and information on computing time for the benchmark instance, we cannot compare time with these algorithms, and give only results of tests from their papers and our experiments so that the reader can have a baseline for these algorithms.

Of the 76 test instances solved with FWLS, there are 45 instances with exact χ for which FWLS is able to find the corresponding optima except graphs of "flat1000_76_0", "le450_15a", "le450_15b", "le450_25c"and "le450_25d", and we list the current optimal solution for other instances found by other algorithms. One thing to be noted that FWLS finds the optima for "flat1000_28_0"using 28 colors, which is the fourth algorithm to solve this special graph [8, 30, 31].

Table 1. Comparison of four algorithms

Graph	$\|V\|$	χ, k^*	DS	ES	AMA	FWLS	#suc	avg time(sec)
DSJC125.1	125	?,5	6	6	**5**	**5**	19/20	0
DSJC125.5	125	?,17	21	21	**17**	**17**	20/20	4.05
DSJC125.9	125	?,44	50	48	**44**	45	7/20	0.05
DSJC250.1	250	?,8	10	10	**8**	**8**	20/20	0.3
DSJC250.5	250	?,28	38	36	**28**	29	19/20	3.21
DSJC250.9	250	?,72	91	82	**72**	73	2/20	1
DSJC500.1	500	?,12	16	15	**12**	13	20/20	0.1
DSJC500.5	500	?,48	67	61	**48**	51	11/20	2116.27
DSJC500.9	500	?,126	161	156	**126**	128	1/20	243
DSJC1000.1	1000	?,20	26	26	**20**	21	1/20	1046
DSJC1000.5	1000	?,83	114	111	84	92	14/20	2141.64
DSJC1000.9	1000	?,222	297	289	224	227	1/20	1034

Table 1. (*continued*)

Graph	$\|V\|$	χ, k^*	DS	ES	AMA	FWLS	#suc	avg time(sec)
flat300_26_0	300	26,26	41	-	**26**	**26**	20/20	0.15
flat300_28_0	300	28,28	41	-	31	**28**	4/20	2627
flat1000_50_0	1000	50,50	112	-	**50**	**50**	20/20	3.65
flat1000_60_0	1000	60,60	113	-	**60**	**60**	20/20	11.25
flat1000_76_0	1000	76,82	114	-	84	90	1/20	1450
fpsol2.i.1	496	65,65	65	-	**65**	**65**	20/20	0
fpsol2.i.2	451	30,30	30	-	**30**	**30**	20/20	0
fpsol2.i.3	425	30,30	30	-	**30**	**30**	20/20	0
inithx.i.1	864	54,54	54	-	**54**	**54**	20/20	0
inithx.i.2	645	31,31	31	-	**31**	**31**	20/20	0
inithx.i.3	621	31,31	31	-	**31**	**31**	20/20	0
le450_5a	450	5,5	10	**5**	**5**	**5**	20/20	0.05
le450_5b	450	5,5	9	**5**	**5**	**5**	20/20	0.15
le450_5c	450	5,5	6	**5**	**5**	**5**	20/20	0
le450_5d	450	5,5	11	**5**	**5**	**5**	20/20	0
le450_15a	450	15,15	16	18	**15**	16	20/20	0.05
le450_15b	450	15,15	16	18	**15**	16	19/20	0.37
le450_15c	450	15,15	24	25	**15**	**15**	20/20	0.2
le450_15d	450	15,15	24	26	**15**	**15**	20/20	0.2
le450_25a	450	25,25	25	26	**25**	**25**	18/20	0.05
le450_25b	450	25,25	25	26	**25**	**25**	18/20	0
le450_25c	450	25,25	29	32	26	26	1/20	3615
le450_25d	450	25,25	28	31	26	26	4/20	3049.75
mugg88_1	88	4,4	**4**	-	**4**	**4**	20/20	0
mugg88_25	88	4,4	**4**	-	**4**	**4**	20/20	0
mugg100_1	100	4,4	**4**	-	**4**	**4**	20/20	0
mugg100_25	100	4,4	**4**	**4**	**4**	**4**	20/20	0
mulsol.i.1	49	49,49	**49**	-	**49**	**49**	20/20	0
mulsol.i.2	31	31,31	**31**	-	**31**	**31**	20/20	0
mulsol.i.3	31	31,31	**31**	-	**31**	**31**	20/20	0
mulsol.i.4	31	31,31	**31**	-	**31**	**31**	20/20	0
mulsol.i.5	31	31,31	**31**	-	**31**	**31**	20/20	0
myciel3	11	4,4	**4**	-	**4**	**4**	20/20	0
myciel4	23	5,5	**5**	-	**5**	**5**	20/20	0
myciel5	47	6,6	**6**	**6**	**6**	**6**	20/20	0
myciel6	95	7,7	**7**	**7**	**7**	**7**	20/20	0
myciel7	191	8,8	**8**	**8**	**8**	**8**	20/20	0
qg.order30	900	30,30	-	-	-	**30**	20/20	0
qg.order40	1600	40,40	-	-	-	**40**	20/20	0.1
qg.order60	3600	60,60	62	-	-	**60**	20/20	0
qg.order100	10000	100,100	103	-	**100**	**100**	20/20	68
queen5_5	25	?,5	5	-	**5**	**5**	20/20	0
queen6_6	36	?,7	9	-	**7**	**7**	20/20	0

Table 1. (*continued*)

Graph	$\|V\|$	χ, k^*	DS	ES	AMA	FWLS	#suc	avg time(sec)
queen7_7	49	?,7	10	-	**7**	**7**	20/20	0
queen8_8	64	?,9	12	10	**9**	**9**	20/20	0.1
queen8_12	96	?,12	13	13	**12**	**12**	20/20	0
queen9_9	81	?,10	14	11	**10**	**10**	20/20	0
queen10_10	100	?,11	13	13	**11**	**11**	19/20	0.21
queen11_11	121	?,11	15	14	**11**	12	20/20	0.5
queen12_12	144	?,12	15	15	13	13	20/20	2.8
queen13_13	169	?,13	17	16	14	14	20/20	0.8
queen14_14	196	?,14	18	17	15	15	20/20	699.9
queen15_15	225	?,15	19	19	16	16	13/20	301.69
queen16_16	256	?,16	21	19	17	18	20/20	0.05
school1	385	14,14	17	-	**14**	**14**	20/20	0
school1_nsh	352	?,14	25	**14**	**14**	**14**	20/20	0
wap01a	2368	?,?	46	-	45	**43**	4/20	810.75
wap02a	2464	?,?	45	-	44	**43**	9/20	891.44
wap03a	4730	?,?	54	-	53	**46**	1/20	266
wap04a	5231	?,?	48	-	**48**	**48**	6/20	3.67
will199GPIA	701	?,7	**7**	-	**7**	**7**	20/20	0
zeroin.i.1	211	49,49	**49**	-	**49**	**49**	20/20	0
zeroin.i.2	211	30,30	**30**	-	**30**	**30**	20/20	0
zeroin.i.3	206	30,30	**30**	-	**30**	**30**	20/20	0

FWLS gets either the optima or the current optimal solution, or outperform the three algorithms for 56 instances, and they are highlighted in bold in Table 1. There exist some instances for which FWLS cannot seek out optima, but find solutions extremely close to the corresponding optima or the current solution within a second. And for the remaining instances, FWLS obtains worse solutions than the optima or the current solutions.

From the viewpoint of instance type, FWLS can obtain significantly good performance on series of "fpsol", "inithx", "mugg", "mulsol", "myciel", "zeroin" and "qg.order" as well as an instance "school1" for which our algorithm can find the corresponding optima in one hundred percent accuracy within a second (avg time is 0 in Table 1 because the machine runs quickly and cannot recognize error). Additionally, FWLS is also able to have good effect on series of "flat", "le", "queen" and "wap" plus an instance "school1_nsh", and it finds the current optimal solutions for most of them. Nevertheless, FWLS cannot be appropriate for DSJC graphs, especially the large and dense instances. There exist relatively large gaps between the current optimal solutions and ones found by FWLS, and additionally, the success rate and run time are not good.

Shown from this table, we can see that FWLS outperforms DS and ES, especially on DSJC graphs. As the number of nodes increases, the effect of FWLS goes evidently. Although FWLS performs worse than AMA on DSJC graphs, it performs best on flat300_28_0 and wap graphs.

5 Conclusion and Future Work

We propose a new LS algorithm named FWLS which resembles WalkSAT, but differs in the tie-breaking mechanism. While WalkSAT breaks ties randomly, FWLS does it by preferring the least recent uncolored vertex. Our algorithm focuses on the uncolored vertices, aggregating them in a set called *conflict set*. Each uncolored vertex contains *age* used to record experience of time in *conflict set*, and a tuple $<v, color, conflictNum>$ which greedily stores the current optimal color and the number of corresponding conflicts between itself and vertices with this optimal color. Our experiments show that FWLS obtains better results than some other heuristic and metaheuristic algorithms, finding the known or current optima for most of the instances.

For future work, some further experiments will be conducted to analyze the differences of performance between FWLS and WalkSAT in depth. For proving the importance of tie-breaking mechanism, we would like to test more mechanisms on all graphs from DIMACS. On the other hand, we would also like to improve FWLS by taking dynamic *noise* in accordance with the present situation of *conflict set* [13–16], and we will take an attempt to other methods such as the *Novelty* families, which are evolved from WalkSAT [15].

Acknowledgements. This work was partially supported by National Basic Research Program (973 Program) 2010CB328103, ARC grants FT0991785, and National Natural Science Foundation of China (61073033, 61003056 and 60903054).

References

1. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness. W.H.Freeman and Company (1979)
2. Chaitin, G.J.: Register allocation and spilling via graph coloring (with retrospective). Best of PLDI, 66–74 (1982)
3. de Werra, D., Eisenbeis, C., Lelait, S., Marmol, B.: On a Graph-theoretical Model for Cyclic Register Allocation. Discrete Applied Mathematics, 191–203 (1999)
4. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyperheuristic for educational timetabling problems. European Journal of Operational Research, 177–192 (2007)
5. Zufferey, N., Amstutz, P., Giaccari, P.: Graph colouring approaches for a satellite range scheduling problem. J. Scheduling, 263–277 (2008)
6. Class, C.: Bag rationalisation for a food manufacture. Journal of the Operational Research Society 5, 544–551 (2002)
7. Hertz, A., de Werra, D.: Using Tabu Search Techniques for Graph Coloring. Computing, 345–351 (1987)
8. Blchliger, I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. Computers & OR, 960–975 (2008)

9. Brélaz, D., Polytechnique, É., de Lausanne, F.: New Methods to Color the Vertices of a Graph. Communications of ACM 22, 251–256 (1979)
10. Kroc, L., Sabharwal, A., Selman, B.: An Empirical Study of Optimal Noise and Runtime Distributions in Local Search. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 346–351. Springer, Heidelberg (2010)
11. Trick, M.A., Yildiz, H.: A Large Neighborhood Search Heuristic for Graph Coloring. In: Van Hentenryck, P., Wolsey, L.A. (eds.) CPAIOR 2007. LNCS, vol. 4510, pp. 346–360. Springer, Heidelberg (2007)
12. Galinier, P., Hertz, A., Zufferey, N.: An adaptive memory algorithm for the k-coloring problem. Discrete Applied Mathematics, 267–279 (2008)
13. Hoos, H.H.: An Adaptive Noise Mechanism for WalkSAT. In: AAAI/IAAI, pp. 655–660 (2002)
14. Balint, A., Fröhlich, A.: Improving Stochastic Local Search for SAT with a New Probability Distribution. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 10–15. Springer, Heidelberg (2010)
15. McAllester, D.A., Selman, B., Kautz, H.A.: Evidence for Invariants in Local Search. In: AAAI/IAAI, pp. 321–326 (1997)
16. Li, C.M., Wei, W., Zhang, H.: Combining Adaptive Noise and Look-Ahead in Local Search for SAT. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 121–133. Springer, Heidelberg (2007)
17. Bui, T.N., Nguyen, T.H., Patel, C.M., Phan, K.-A.T.: An ant-based algorithm for coloring graphs. Discrete Applied Mathematics, 190–200 (2008)
18. Lü, Z., Hao, J.-K.: A memetic algorithm for graph coloring. European Journal of Operational Research, 241–250 (2010)
19. Malaguti, E., Monaci, M., Toth, P.: A Metaheuristic Approach for the Vertex Coloring Problem. INFORMS Journal on Computing, 302–316 (2008)
20. Porumbel, D.C., Hao, J.-K., Kuntz, P.: An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Computers & OR, 1822–1832 (2010)
21. Hamiez, J.-P., Hao, J.-K.: Scatter Search for Graph Coloring. In: Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E., Schoenauer, M. (eds.) EA 2001. LNCS, vol. 2310, pp. 168–179. Springer, Heidelberg (2002)
22. Fridcn, C., Hertz, A., de Werra, D.: Lausanne: STABULUS: A Technique for Finding Stable Sets in Large Graphs with Tabu Search Computing. 42, 35–44 (1989)
23. Yesil, C., Yilmaz, B., Korkmaz, E.E.: Hybrid local search algorithms on Graph Coloring Problem. In: HIS, pp. 468–473 (2011)
24. Galinier, P., Hao, J.-K.: Hybrid Evolutionary Algorithms for Graph Coloring. J. Comb. Optim., 379–397 (1999)
25. Gent, I.P., Walsh, T.: Towards an Understanding of Hill-Climbing Procedures for SAT. In: AAAI, pp. 28–33 (1993)
26. Garey, M.R., Johnson, D.S.: In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability (2009)
27. Bounkong, S., van Mourik, J., Saad, D.: Coloring random graphs and maximizing local diversity. Physical Review E (2006)
28. Galinier, P., Hertz, A.: A survey of local search methods for graph coloring. Computers & Operations Research 33, 2547–2562 (2006)
29. Leighton, F.T.: A Graph Coloring Algorithm for Large Scheduling Problems. Journal of Research of the National Bureau of Standards 84, 489–506 (1979)
30. Porumbel, D.C., Hao, J.-K., Kuntz, P.: A search space "cartography" for guiding graph coloring heuristics. Computers & OR, 769–778 (2010)
31. Hertz, A., Plumettaz, M., Zufferey, N.: Variable space search for graph coloring. Discrete Applied Mathematics, 2551–2560 (2008)

A One-Vertex Decomposition Algorithm for Generating Algebraic Expressions of Square Rhomboids

Mark Korenblit[1] and Vadim E. Levit[2]

[1] Holon Institute of Technology, Israel
korenblit@hit.ac.il
[2] Ariel University, Israel
levitv@ariel.ac.il

Abstract. The paper investigates relationship between algebraic expressions and graphs. We consider a digraph called a square rhomboid that is an example of non-series-parallel graphs. Our intention is to simplify the expressions of square rhomboids and eventually find their shortest representations. With that end in view, we describe the new algorithm for generating square rhomboid expressions based on the decomposition method.

1 Introduction

A *graph* $G = (V, E)$ consists of a *vertex set* V and an *edge set* E, where each edge corresponds to a pair (v, w) of vertices. A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A graph G is a *homeomorph* of G' if G can be obtained by subdividing edges of G' with new vertices. We say that a graph $G^2 = (V, E')$ is a *square of a graph* $G = (V, E)$ if $E' = \{(u, w) : (u, w) \in E \vee ((u, v) \in E \wedge (v, w) \in E)$ for some $v \in V\}$. A two-terminal directed acyclic graph (*st-dag*) has only one source and only one sink.

We consider a *labeled graph* which has labels attached to its edges. Each path between the source and the sink (a *sequential path*) in an st-dag can be presented by a product of all edge labels of the path. We define the sum of edge label products corresponding to all possible sequential paths of an st-dag G as the *canonical expression* of G. An algebraic expression is called an *st-dag expression* (a *factoring of an st-dag* in [2]) if it is algebraically equivalent to the canonical expression of an st-dag. An st-dag expression consists of literals (edge labels), and the operators + (disjoint union) and · (concatenation, also denoted by juxtaposition). An expression of an st-dag G will be hereafter denoted by $Ex(G)$.

We define the total number of literals in an algebraic expression as the *complexity of the algebraic expression*. An equivalent expression with the minimum complexity is called an *optimal representation of the algebraic expression*. We consider expressions with a minimum (or, at least, a polynomial) complexity as a key to generating efficient algorithms on distributed systems.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 94–105, 2013.

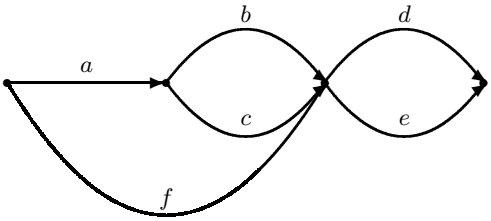


Fig. 1. A series-parallel graph

A *series-parallel graph* is defined recursively so that a single edge is a series-parallel graph and a graph obtained by a parallel or a series composition of series-parallel graphs is series-parallel. As shown in [2] and [10], a series-parallel graph expression has a representation in which each literal appears only once. This representation is an optimal representation of the series-parallel graph expression. For example, the canonical expression of the series-parallel graph presented in Figure 1 is $abd + abe + acd + ace + fe + fd$. Since it is a series-parallel graph, the expression can be reduced to $(a(b + c) + f)(d + e)$, where each literal appears once.

A *Fibonacci graph* [7] has vertices $\{1, 2, 3, \ldots, n\}$ and edges $\{(v, v + 1) \mid v = 1, 2, \ldots, n - 1\} \cup \{(v, v + 2) \mid v = 1, 2, \ldots, n - 2\}$. As shown in [4], an st-dag is series-parallel if and only if it does not contain a subgraph which is a homeomorph of the *forbidden subgraph* positioned between vertices 1 and 4 of the Fibonacci graph illustrated in Figure 2. Thus a Fibonacci graph gives a generic example of non-series-parallel graphs.

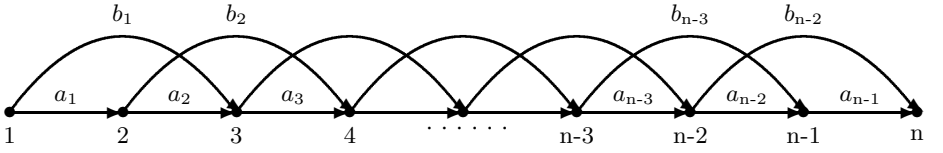


Fig. 2. A Fibonacci graph

Mutual relations between graphs and expressions are discussed in [2], [5], [6], [10], [11], [12], [13], [14], [15], [17], and other works. Specifically, [13], [14], and [17] consider the correspondence between series-parallel graphs and read-once functions. A Boolean function is defined as *read-once* if it may be computed by some formula in which no variable occurs more than once (*read-once formula*). On the other hand, a series-parallel graph expression can be reduced to the representation in which each literal appears only once. Hence, such a representation of a series-parallel graph expression can be considered as a read-once formula (boolean operations are replaced by arithmetic ones).

An expression of a homeomorph of the forbidden subgraph belonging to any non-series-parallel st-dag has no representation in which each literal appears once. For example, consider the subgraph positioned between vertices 1 and 4 of

the Fibonacci graph shown in Figure 2. Possible optimal representations of its expression are $a_1\,(a_2a_3 + b_2) + b_1a_3$ or $(a_1a_2 + b_1)\,a_3 + a_1b_2$. For this reason, an expression of a non-series-parallel st-dag can not be represented as a read-once formula. However, for arbitrary functions, which are not read-once, generating the optimum factored form is NP-complete [18]. Some algorithms developed in order to obtain good factored forms are described in [5], [6] and other works. In [10] we presented an algorithm, which generates the expression of $O\left(n^2\right)$ complexity for an n-vertex Fibonacci graph.

There are many practical applications with *planar graphs*, i.e., the graphs that can be drawn in the plane without any edges crossing. In this paper we investigate a non-series-parallel st-dag called a *square rhomboid* (Figure 3). This graph looks like a planar approximation of the square of a *rhomboid*, which is a series composition of *rhomb* graphs. A square rhomboid consists of the same vertices as the corresponding rhomboid. However, edges labeled by letters a, b, and c (see Figure 3) are absent in a rhomboid. Geometrically, a square rhomboid can be considered to be a "gluing" of two Fibonacci graphs, i.e., it is the next harder one in a sequence of increasingly non-series-parallel graphs.

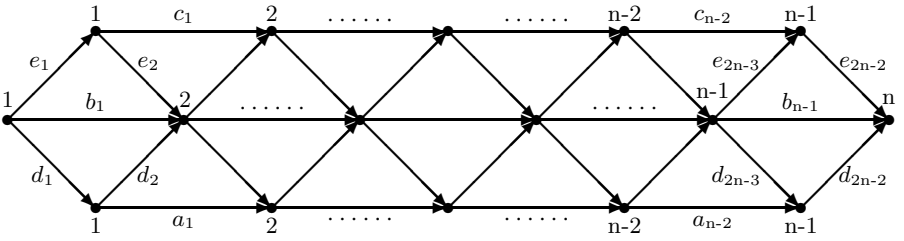


Fig. 3. A square rhomboid of size n

The set of vertices of an N-vertex square rhomboid consists of $\frac{N+2}{3}$ *middle* (*basic*), $\frac{N-1}{3}$ *upper*, and $\frac{N-1}{3}$ *lower* vertices. Upper and lower vertices numbered x will be denoted in formulae by $\overline{x}$ and $\underline{x}$, respectively. The square rhomboid (SR for brevity) including n basic vertices will be denoted by $SR(n)$ and will be called an SR of size n.

Our intention in this paper is to generate and to simplify the expressions of square rhomboids. With that end in view, we present an algorithm based on a *decomposition method*.

2 A One-Vertex Decomposition Method (1-VDM)

The method is based on revealing subgraphs in the initial graph. The resulting expression is produced by a special composition of subexpressions describing these subgraphs.

For a non-trivial SR subgraph with a source p and a sink q we choose any *decomposition vertex* i ($p < i < q$) located in the basic group of a subgraph.

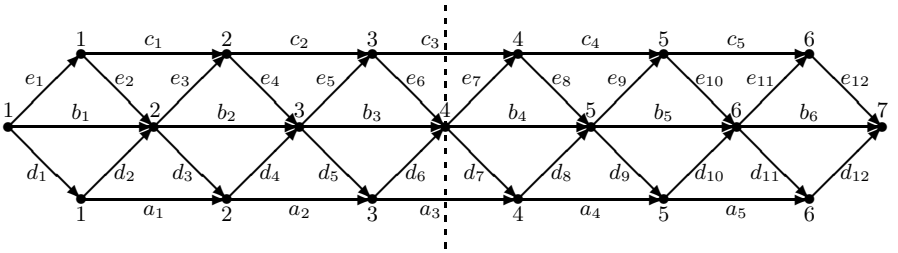


Fig. 4. Decomposition of a square rhomboid of size 7 at vertex 4

We conditionally split each SR through its decomposition vertex (see the example in Figure 4).

Two kinds of subgraphs are revealed in the graph in the course of decomposition. The first of them is an SR with a fewer number of vertices than the initial SR. The second one is an SR supplemented by two additional edges at one of four sides. Possible varieties of this st-dag (we call it a *single-leaf square rhomboid* and denote by $\widehat{SR}$) which are subgraphs revealed from an SR in Figure 4, are illustrated in Figure 5(a, b, c, d). Let $\widehat{SR}(n)$ (an $\widehat{SR}$ of size n) denote an $\widehat{SR}$ including n basic vertices.

We denote by $E(p, q)$ a subexpression related to an SR subgraph with a source p and a sink q. We denote by $E(p, \overline{q})$, $E(\overline{p}, q)$, $E(p, \underline{q})$, $E(\underline{p}, q)$ subexpressions related to $\widehat{SR}$ subgraphs of Figure 5(a, b, c, d, respectively) with a source p and a sink $\overline{q}$, a source $\overline{p}$ and a sink q, a source p and a sink $\underline{q}$, and a source $\underline{p}$ and a sink q.

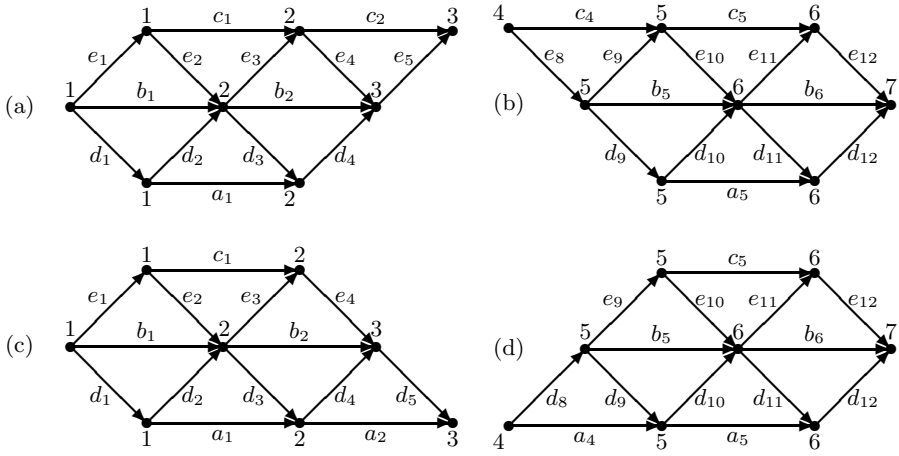


Fig. 5. Varieties of a single-leaf square rhomboid of size 3

Any path from vertex 1 to vertex 7 in Figure 4 passes through decomposition vertex 4 or through edge c_3 or through edge a_3. Therefore, $E(p, q)$ is generated by the following recursive procedure (*decomposition procedure*):

1. **case** $q = p : E(p, q) \leftarrow 1$
2. **case** $q = p + 1 : E(p, q) \leftarrow b_p + e_{2p-1}e_{2p} + d_{2p-1}d_{2p}$
3. **case** $q > p + 1 : \mathbf{choice}(p, q, i)$
4. $E(p, q) \leftarrow E(p, i)E(i, q) + E(p, \overline{i-1})c_{i-1}E(\overline{i}, q) + E(p, \underline{i-1})a_{i-1}E(\underline{i}, q)$

The expression related to a one-vertex SR is defined formally as 1 (line 1). Line 2 describes the expression of $SR(2)$ which is a trivial subgraph. The procedure $\mathbf{choice}(p, q, i)$ in line 3 chooses an arbitrary number i on the interval (p, q) so that $p < i < q$. This number is a number of a decomposition vertex in a basic group of the SR subgraph positioned between vertices p and q. A current subgraph is decomposed into six new subgraphs in line 4. Subgraphs described by subexpressions $E(p, i)$ and $E(i, q)$ include all paths from vertex p to vertex q passing through vertex i. Subgraphs described by subexpressions $E(p, \overline{i-1})$ and $E(\overline{i}, q)$ include all paths from vertex p to vertex q passing via edge c_{i-1}. Subgraphs described by subexpressions $E(p, \underline{i-1})$ and $E(\underline{i}, q)$ include all paths from vertex p to vertex q passing via edge a_{i-1}.

$E(1, n)$ is the expression of the initial SR of size n. Hence, the decomposition procedure is initially invoked by substituting 1 and n instead of p and q, respectively.

We now consider the algorithm that generates the algebraic expression for a square rhomboid using 1-VDM as its base.

3 A One-Vertex Decomposition Algorithm (1-VDA)

The algorithm is based on decomposition of the initial graph and all kinds of subgraphs to new subgraphs.

We conjecture that the representation with the minimum complexity of $Ex(SR)$ derived by 1-VDM is generated when the decomposition vertex is a middle vertex in the basic group of the SR. That is, the number i of the decomposition vertex for a current SR subgraph which is positioned between vertices p and q is chosen as $\frac{q+p}{2}$ ($\lceil \frac{q+p}{2} \rceil$ or $\lfloor \frac{q+p}{2} \rfloor$). In other words, i is equal to $\frac{q+p}{2}$ for odd $q - p + 1$ and i equals $\frac{q+p-1}{2}$ or $\frac{q+p+1}{2}$ for even $q - p + 1$.

An $\widehat{SR}$ subgraph is decomposed through a decomposition vertex selected in its basic group into six new subgraphs in the same way as an SR (see the examples in Figure 6). The decomposition vertex is chosen so that the location of the split is in the middle of the subgraph.

Three kinds of subgraphs are revealed in an $\widehat{SR}$ in the course of decomposition. The first and the second of them are an SR and an $\widehat{SR}$, respectively. The third one is an SR supplemented by two additional pairs of edges (one pair is on the left and another one is on the right). Possible varieties of this st-dag (we call it a *dipterous square rhomboid* and denote it by $\widehat{\widehat{SR}}$) are illustrated in Figure 7(a, b, c, d). We define subgraphs illustrated in Figure 7(a, c) as *trapezoidal* $\widehat{\widehat{SR}}$

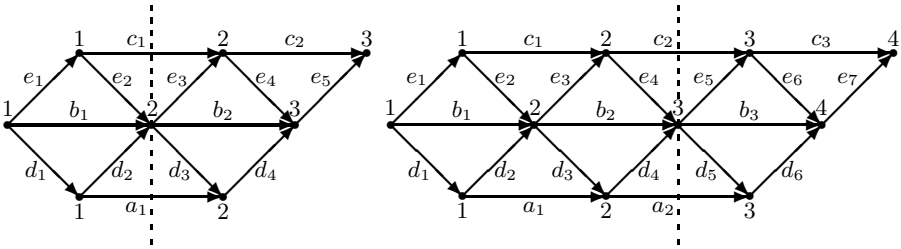


Fig. 6. Decomposition of single-leaf square rhomboids by 1-VDA

graphs and subgraphs illustrated in Figure 7(b, d) as *parallelogram* $\widehat{\widehat{SR}}$ *graphs*. Let $\widehat{\widehat{SR}}(n)$ (an $\widehat{\widehat{SR}}$ of size n) denote an $\widehat{\widehat{SR}}$ including n basic vertices.

We denote by $E(\overline{p},\overline{q})$, $E(\underline{p},\overline{q})$, $E(\underline{p},\underline{q})$, $E(\overline{p},\underline{q})$ subexpressions related to subgraphs of Figure 7(a, b, c, d, respectively) with a source $\overline{p}$ and a sink $\overline{q}$, a source $\underline{p}$ and a sink $\overline{q}$, a source $\underline{p}$ and a sink $\underline{q}$, and a source $\overline{p}$ and a sink $\underline{q}$.

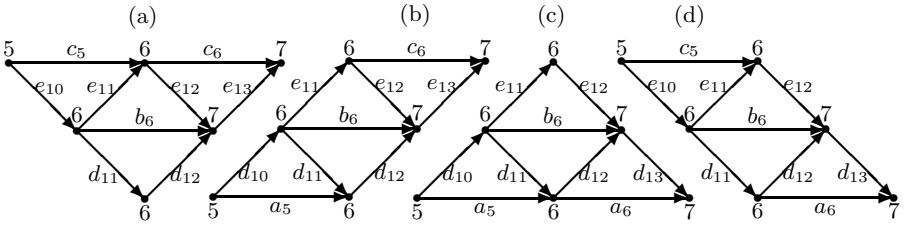


Fig. 7. Varieties of a dipterous square rhomboid of size 2

An $\widehat{\widehat{SR}}$ subgraph is decomposed into six new subgraphs in the same way as an SR and an $\widehat{SR}$ (see the examples in Figure 8). The number i of the decomposition vertex in the basic group for a current $\widehat{\widehat{SR}}$ subgraph which is positioned between vertices p and q, is chosen as $\frac{q+p+1}{2}$ ($\lceil\frac{q+p+1}{2}\rceil$ or $\lfloor\frac{q+p+1}{2}\rfloor$).

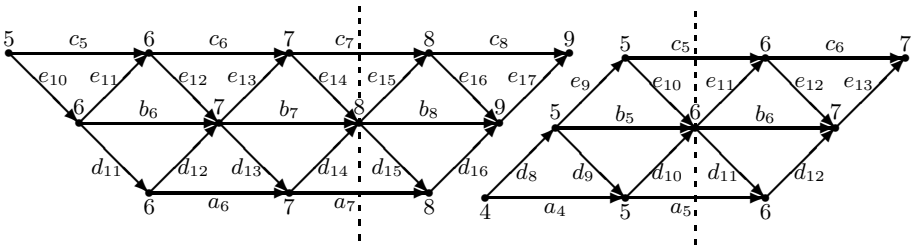


Fig. 8. Decomposition of dipterous square rhomboids by 1-VDA

In the course of decomposition, two kinds of subgraphs are revealed in an $\widehat{\widehat{SR}}$. They are an $\widehat{SR}$ and an $\widehat{\widehat{SR}}$. Therefore, no new kinds of subgraphs are formed and subexpressions related to all subgraphs can be generated.

Hence, $Ex(SR)$ is computed by the following recursive relations:

1. $E(p,p) = 1$
2. $E(p,\overline{p}) = e_{2p-1}$ 3. $E(p,\underline{p}) = d_{2p-1}$
4. $E(\overline{p},p+1) = e_{2p}$ 5. $E(\underline{p},p+1) = d_{2p}$
6. $E(\overline{p},\overline{p+1}) = c_p + e_{2p}e_{2p+1}$ 7. $E(\overline{p},\underline{p+1}) = e_{2p}d_{2p+1}$
8. $E(\underline{p},\overline{p+1}) = d_{2p}e_{2p+1}$ 9. $E(\underline{p},\underline{p+1}) = a_p + d_{2p}d_{2p+1}$
10. $E(p,p+1) = b_p + e_{2p-1}e_{2p} + d_{2p-1}d_{2p}$
11. $E(p,\overline{p+1}) = (b_p + d_{2p-1}d_{2p})e_{2p+1} + e_{2p-1}(c_p + e_{2p}e_{2p+1})$
12. $E(p,\underline{p+1}) = (b_p + e_{2p-1}e_{2p})d_{2p+1} + d_{2p-1}(a_p + d_{2p}d_{2p+1})$
13. $E(\overline{p},p+2) = (c_p + e_{2p}e_{2p+1})e_{2p+2} + e_{2p}(b_{p+1} + d_{2p+1}d_{2p+2})$
14. $E(\underline{p},p+2) = (a_p + d_{2p}d_{2p+1})d_{2p+2} + d_{2p}(b_{p+1} + e_{2p+1}e_{2p+2})$
15. $E(\overline{p},\overline{p+2}) = e_{2p}(b_{p+1} + d_{2p+1}d_{2p+2})e_{2p+3} +$
$(a_p + d_{2p}d_{2p+1})(a_{p+1} + d_{2p+2}d_{2p+3})$
16. $E(\overline{p},\underline{p+2}) = e_{2p}(b_{p+1}d_{2p+3} + d_{2p+1}(a_{p+1} + d_{2p+2}d_{2p+3})) +$
$(c_p + e_{2p}e_{2p+1})e_{2p+2}d_{2p+3}$
17. $E(\underline{p},\overline{p+2}) = d_{2p}(b_{p+1}e_{2p+3} + e_{2p+1}(c_{p+1} + e_{2p+2}e_{2p+3})) +$
$(a_p + d_{2p}d_{2p+1})d_{2p+2}e_{2p+3}$
18. $E(\underline{p},\underline{p+2}) = d_{2p}(b_{p+1} + e_{2p+1}e_{2p+2})d_{2p+3} +$
$(c_p + e_{2p}e_{2p+1})(\underline{c_{p+1}} + e_{2p+2}e_{2p+3})$
19. $E(p,\overline{q}) = E(p,i)E(i,\overline{q}) + E(p,\overline{i-1})c_{i-1}E(\overline{i},\overline{q}) + E(p,\underline{i-1})a_{i-1}E(\underline{i},\overline{q})$,
$i = \left\lceil \frac{q+p}{2} \right\rceil$ $(q > p+1)$
20. $E(p,\underline{q}) = E(p,i)E(i,\underline{q}) + E(p,\overline{i-1})c_{i-1}E(\overline{i},\underline{q}) + E(p,\underline{i-1})a_{i-1}E(\underline{i},\underline{q})$,
$i = \left\lceil \frac{q+p}{2} \right\rceil$ $(q > p+1)$
21. $E(\overline{p},q) = E(\overline{p},i)E(i,q) + E(\overline{p},\overline{i-1})c_{i-1}E(\overline{i},q) + E(\overline{p},\underline{i-1})a_{i-1}E(\underline{i},q)$,
$i = \left\lceil \frac{q+p}{2} \right\rceil$ $(q > p+2)$
22. $E(\underline{p},q) = E(\underline{p},i)E(i,q) + E(\underline{p},\overline{i-1})c_{i-1}E(\overline{i},q) + E(\underline{p},\underline{i-1})a_{i-1}E(\underline{i},q)$,
$i = \left\lceil \frac{q+p}{2} \right\rceil$ $(q > p+2)$
23. $E(\overline{p},\overline{q}) = E(\overline{p},i)E(i,\overline{q}) + E(\overline{p},\overline{i-1})c_{i-1}E(\overline{i},\overline{q}) + E(\overline{p},\underline{i-1})a_{i-1}E(\underline{i},\overline{q})$,
$i = \frac{q+p+1}{2}$ $(q > p+2)$
24. $E(\overline{p},\underline{q}) = E(\overline{p},i)E(i,\underline{q}) + E(\overline{p},\overline{i-1})c_{i-1}E(\overline{i},\underline{q}) + E(\overline{p},\underline{i-1})a_{i-1}E(\underline{i},\underline{q})$,
$i = \frac{q+p+1}{2}$ $(q > p+2)$
25. $E(\underline{p},\overline{q}) = E(\underline{p},i)E(i,\overline{q}) + E(\underline{p},\overline{i-1})c_{i-1}E(\overline{i},\overline{q}) + E(\underline{p},\underline{i-1})a_{i-1}E(\underline{i},\overline{q})$,
$i = \frac{q+p+1}{2}$ $(q > p+2)$
26. $E(\underline{p},\underline{q}) = E(\underline{p},i)E(i,\underline{q}) + E(\underline{p},\overline{i-1})c_{i-1}E(\overline{i},\underline{q}) + E(\underline{p},\underline{i-1})a_{i-1}E(\underline{i},\underline{q})$,
$i = \frac{q+p+1}{2}$ $(q > p+2)$
27. $E(p,q) = E(p,i)E(i,q) + E(p,\overline{i-1})c_{i-1}E(\overline{i},q) + E(p,\underline{i-1})a_{i-1}E(\underline{i},q)$,
$i = \frac{q+p}{2}$ $(q > p+1)$.

Subgraphs of sizes 1 and 2 are trivial (their expressions are in lines 1 – 18). Specifically, expressions of subgraphs $\widehat{SR}(2)$ and $\widehat{\widehat{SR}}(2)$ are presented in the minimum factored form (lines 11 – 18).

For example, the expression of the square rhomboid of size 3 derived by 1-VDA is

$$(b_1 + e_1e_2 + d_1d_2)(b_2 + e_3e_4 + d_3d_4) + e_1c_1e_4 + d_1a_1d_4.$$

It contains 16 literals.

Lemma 1. *Complexities of expressions* $Ex\left(\textit{trapezoidal } \widehat{\widehat{SR}}(n)\right)$ *and* $Ex\left(\textit{parallelogram } \widehat{\widehat{SR}}(n)\right)$ *derived by 1-VDA are equal for* $n > 2$.

Proof. Expressions $Ex\left(\textit{trapezoidal } \widehat{\widehat{SR}}\,(n)\right)$ and $Ex\left(\textit{parallelogram } \widehat{\widehat{SR}}\,(n)\right)$ consist of the same components (see lines 23 – 26 of 1-VDA) for $n > 2$. They are the literals c_{i-1} and a_{i-1} and the following six subexpressions:

$Ex\left(\widehat{SR}\left(\lceil \frac{n}{2} \rceil\right)\right)$; $Ex\left(\widehat{SR}\left(\lfloor \frac{n}{2} \rfloor + 1\right)\right)$;

$Ex\left(\textit{trapezoidal } \widehat{\widehat{SR}}\left(\lceil \frac{n}{2} \rceil - 1\right)\right)$; $Ex\left(\textit{trapezoidal } \widehat{\widehat{SR}}\left(\lfloor \frac{n}{2} \rfloor\right)\right)$;

$Ex\left(\textit{parallelogram } \widehat{\widehat{SR}}\left(\lceil \frac{n}{2} \rceil - 1\right)\right)$; $Ex\left(\textit{parallelogram } \widehat{\widehat{SR}}\left(\lfloor \frac{n}{2} \rfloor\right)\right)$.

The subexpression of each kind appears once in $Ex\left(\textit{trapezoidal } \widehat{\widehat{SR}}\right)$ and once in $Ex\left(\textit{parallelogram } \widehat{\widehat{SR}}\right)$. Hence, the expression complexity for any $\widehat{\widehat{SR}}\,(n)$ is equal to the sum of complexities of the subexpressions above increased by two. For this reason, complexities of expressions $Ex\left(\textit{trapezoidal } \widehat{\widehat{SR}}\right)$ and $Ex\left(\textit{parallelogram } \widehat{\widehat{SR}}\right)$are equal for $n > 2$. □

Proposition 1. *The total number of literals* $T(n)$ *in the expression* $Ex(SR(n))$ *derived by 1-VDA is defined recursively as follows:*

1) $T(1) = 0$; *2)* $\widehat{T}(1) = 1$; *3)* $\widehat{\widehat{T}}_{pr}(1) = 2$; *4)* $\widehat{\widehat{T}}_{tr}(1) = 3$

5) $T(2) = 5$; *6)*$\widehat{T}(2) = 8$; *7)* $\widehat{\widehat{T}}_{pr}(2) = 12$; *8)* $\widehat{\widehat{T}}_{tr}(2) = 11$

9) $\widehat{T}(3) = 22$; *10)* $\widehat{\widehat{T}}(3) = 28$; *11)* $\widehat{T}(4) = 47$; *12)* $\widehat{\widehat{T}}(4) = 60$

13) $\widehat{T}(5) = 79$; *14)* $\widehat{\widehat{T}}(5) = 92$; *15)* $\widehat{T}(6) = 132$; *16)* $\widehat{\widehat{T}}(6) = 50$

17) $T(n) = T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor + 1\right) + 2\widehat{T}\left(\lceil \frac{n}{2} \rceil - 1\right) + 2\widehat{T}\left(\lfloor \frac{n}{2} \rfloor\right) + 2 \quad (n > 2)$

18) $\widehat{T}(n) = T\left(\lfloor \frac{n}{2} \rfloor + 1\right) + \widehat{T}\left(\lceil \frac{n}{2} \rceil\right) + 2\widehat{T}\left(\lfloor \frac{n}{2} \rfloor\right) + 2\widehat{\widehat{T}}\left(\lceil \frac{n}{2} \rceil - 1\right) + 2 \quad (n > 6)$

19) $\widehat{\widehat{T}}(n) = \widehat{T}\left(\lceil \frac{n}{2} \rceil\right) + \widehat{T}\left(\lfloor \frac{n}{2} \rfloor + 1\right) + 2\widehat{\widehat{T}}\left(\lceil \frac{n}{2} \rceil - 1\right) + 2\widehat{\widehat{T}}\left(\lfloor \frac{n}{2} \rfloor\right) + 2 \quad (n > 6)$,

where $\widehat{T}(n)$, $\widehat{\widehat{T}}_{pr}(n)$, $\widehat{\widehat{T}}_{tr}(n)$, *and* $\widehat{\widehat{T}}(n)$ *are the total numbers of literals in* $Ex(\widehat{SR}(n))$, $Ex\left(\textit{parallelogram } \widehat{\widehat{SR}}\,(n)\right)$, $Ex\left(\textit{trapezoidal } \widehat{\widehat{SR}}\,(n)\right)$, *and* $Ex\left(\widehat{\widehat{SR}}\,(n)\right)$, *respectively.*

Proof. Initial statements (1 – 8) follow directly from lines 1 – 18 of 1-VDA. General statements (17 – 19) are based on the structure of expressions 19 – 27 of 1-VDA and on Lemma 1. The applying of 1-VDA to graphs $\widehat{SR}(3)$, $\widehat{\widehat{SR}}(3)$, $\widehat{SR}(4)$, $\widehat{\widehat{SR}}(4)$, $\widehat{SR}(5)$, $\widehat{\widehat{SR}}(5)$, $\widehat{SR}(6)$, $\widehat{\widehat{SR}}(6)$ leads to the revealing, specifically, of trapezoidal and parallelogram $\widehat{\widehat{SR}}(1)$ and $\widehat{\widehat{SR}}(2)$ subgraphs. For this reason, complexities related to expressions of these graphs are computed exclusively by applying the following formulae which are based on lines 19 – 26 of 1-VDA:

$$\begin{aligned}\widehat{T}(3) &= T(2) + \widehat{T}(2) + 2\widehat{T}(1) + \widehat{\widehat{T}}_{pr}(1) + \widehat{\widehat{T}}_{tr}(1) + 2\\ &= 5 + 8 + 2 + 2 + 3 + 2 = 22\\ \widehat{\widehat{T}}(3) &= 2\widehat{T}(2) + 2\widehat{\widehat{T}}_{pr}(1) + 2\widehat{\widehat{T}}_{tr}(1) + 2 = 16 + 4 + 6 + 2 = 28\end{aligned}$$

$$\begin{aligned}\widehat{T}(4) &= T(3) + 3\widehat{T}(2) + \widehat{\widehat{T}}_{pr}(1) + \widehat{\widehat{T}}_{tr}(1) + 2 = 16 + 24 + 2 + 3 + 2 = 47\\ \widehat{\widehat{T}}(4) &= \widehat{T}(3) + \widehat{T}(2) + \widehat{\widehat{T}}_{pr}(2) + \widehat{\widehat{T}}_{pr}(1) + \widehat{\widehat{T}}_{tr}(2) + \widehat{\widehat{T}}_{tr}(1) + 2\\ &= 22 + 8 + 12 + 2 + 11 + 3 + 2 = 60\\ \widehat{T}(5) &= T(3) + \widehat{T}(3) + 2\widehat{T}(2) + \widehat{\widehat{T}}_{pr}(2) + \widehat{\widehat{T}}_{tr}(2) + 2\\ &= 16 + 22 + 16 + 12 + 11 + 2 = 79\\ \widehat{\widehat{T}}(5) &= 2\widehat{T}(3) + 2\widehat{\widehat{T}}_{pr}(2) + 2\widehat{\widehat{T}}_{tr}(2) + 2 = 44 + 24 + 22 + 2 = 92\\ \widehat{T}(6) &= T(4) + 3\widehat{T}(3) + \widehat{\widehat{T}}_{pr}(2) + \widehat{\widehat{T}}_{tr}(2) + 2 = 41 + 66 + 12 + 11 + 2 = 132\\ \widehat{\widehat{T}}(6) &= \widehat{T}(4) + \widehat{T}(3) + 2\widehat{\widehat{T}}(3) + \widehat{\widehat{T}}_{pr}(2) + \widehat{\widehat{T}}_{tr}(2) + 2\\ &= 47 + 22 + 56 + 12 + 11 + 2 = 150.\end{aligned}$$

The obtained results are initial statements (9 – 16). □

Hence, the expression $Ex(SR(n))$ derived by 1-VDA consists of six subexpressions related to six revealed subgraphs of size $n' \approx \frac{n}{2}$ and two additional literals. Each of these subexpressions is constructed in its turn from six subexpressions related to six subgraphs of size $n'' \approx \frac{n}{4}$ and two additional literals, etc. Thus by the master theorem, $T(n) = O\left(n^{\log_2 6}\right)$, i.e., 1-VDA provides the representation of $Ex(SR(n))$ with a polynomial complexity.

It is of interest to obtain exact formulae describing complexity of the expression $Ex(SR(n))$ derived by 1-VDA. We attempt to do it for n that is a power

of two, i.e., $n = 2^k$ for some positive integer $k \geq 2$. Statements (17 – 19) of Proposition 1 are presented for $n = 2^k$ $(k \geq 3)$ as

$$\begin{cases} T(n) = T\left(\frac{n}{2}+1\right) + T\left(\frac{n}{2}\right) + 2\widehat{T}\left(\frac{n}{2}\right) + 2\widehat{T}\left(\frac{n}{2}-1\right) + 2 \\ \widehat{T}(n) = T\left(\frac{n}{2}+1\right) + 3\widehat{T}\left(\frac{n}{2}\right) + 2\widehat{\widehat{T}}\left(\frac{n}{2}-1\right) + 2 \\ \widehat{\widehat{T}}(n) = \widehat{T}\left(\frac{n}{2}+1\right) + \widehat{T}\left(\frac{n}{2}\right) + 2\widehat{\widehat{T}}\left(\frac{n}{2}\right) + 2\widehat{\widehat{T}}\left(\frac{n}{2}-1\right) + 2\ , \end{cases} \tag{1}$$

respectively. After a number of transformations, the following explicit formulae for simultaneous recurrences (1) are obtained by the method for linear recurrence relations solving [16]:

$$\begin{aligned} T(n) &= \frac{154}{135}n^{\log_2 6} + \frac{1}{27}n^{\log_2 3} - \frac{2}{5} \\ \widehat{T}(n) &= \frac{154}{135}n^{\log_2 6} + \frac{19}{27}n^{\log_2 3} - \frac{2}{5} \\ \widehat{\widehat{T}}(n) &= \frac{154}{135}n^{\log_2 6} + \frac{58}{27}n^{\log_2 3} - \frac{2}{5}. \end{aligned}$$

4 Comparison of 1-VDA with Other Algorithms

Another decomposition method (called 2-VDM) for generating algebraic expressions of square rhomboids is presented in [11] and [12]. The method consists in splitting a square rhomboid through two decomposition vertices one of which belongs to the upper group and another one to the lower group. The following algorithms based on this method are considered in [11] and [12]: full decomposition algorithm (FDA), combined decomposition algorithm (CDA), and improved FDA (IFDA). The values of $T(n)$ for these algorithms and for 1-VDA are presented in Table 1. One can see that 1-VDA is significantly more efficient than FDA, CDA, and IFDA.

Table 1. Complexities for FDA, CDA, IFDA, and 1-VDA

n	$T(n)$, FDA	$T(n)$, CDA	$T(n)$, IFDA	$T(n)$, 1-VDA
4	47	43	43	41
5	110	102	100	66
6	173	161	157	119
7	252	236	228	172
8	331	311	299	247
9	520	488	470	322
10	709	665	641	439
20	4527	4283	4071	2675
30	11669	11113	10461	7597
40	27979	26575	25099	16169
50	48733	46429	43585	28741

5 Conclusions and Future Work

The algorithms presented both in [10] and this paper generate expressions of polynomial size for Fibonacci graphs and square rhomboids, respectively. These algorithms are based on decomposition methods. The existence of a decomposition method for a graph G is a sufficient condition for the existence of an expression with polynomial complexity for G. The complexity depends, in particular, on the number of revealed subgraphs in each recursive step of the decomposition procedure.

A graph in which every subgraph has a vertex of degree at most k is called *k-inductive* [8], for instance, trees, planar graphs, etc. *Random scale-free networks* demonstrate important practical examples of these graphs [1]. The *linkage of a graph* is the smallest value of k for which it is k-inductive [9]. As follows from [3], a graph G is k-inductive if and only if the edges of G can be oriented to form a directed acyclic graph with out-degree at most k. Thus underlying graphs of Fibonacci graphs and square rhomboids are k-inductive graphs with linkages 2 and 3, respectively.

Our intent is to extend the presented decomposition technique to a class of st-dags whose underlying graphs are k-inductive.

References

1. Barabási, A.-L., Albert, R.: Emergence of Scaling in Random Networks. Science 286(5439), 509–512 (1999)
2. Bein, W.W., Kamburowski, J., Stallmann, M.F.M.: Optimal Reduction of Two-Terminal Directed Acyclic Graphs. SIAM Journal of Computing 21(6), 1112–1129 (1992)
3. Chrobak, M., Eppstein, D.: Planar Orientations with Low Out-Degree and Compaction of Adjacency Matrices. Theoretical Computer Science 86(2), 243–266 (1991)
4. Duffin, R.J.: Topology of Series-Parallel Networks. Journal of Mathematical Analysis and Applications 10, 303–318 (1965)
5. Golumbic, M.C., Mintz, A.: Factoring Logic Functions Using Graph Partitioning. In: Proc. IEEE/ACM Int. Conf. Computer Aided Design, pp. 109–114 (1999)
6. Golumbic, M.C., Mintz, A., Rotics, U.: Factoring and Recognition of Read-Once Functions using Cographs and Normality. In: Proc. 38th Design Automation Conf., pp. 195–198 (2001)
7. Golumbic, M.C., Perl, Y.: Generalized Fibonacci Maximum Path Graphs. Discrete Mathematics 28, 237–245 (1979)
8. Irani, S.: Coloring Inductive Graphs On-Line. Algorithmica 11(1), 53–72 (1994)
9. Kirousis, L.M., Thilikos, D.M.: The Linkage of a Graph. SIAM Journal on Computing 25(3), 626–647 (1996)
10. Korenblit, M., Levit, V.E.: On Algebraic Expressions of Series-Parallel and Fibonacci Graphs. In: Calude, C.S., Dinneen, M.J., Vajnovszki, V. (eds.) DMTCS 2003. LNCS, vol. 2731, pp. 215–224. Springer, Heidelberg (2003)
11. Korenblit, M., Levit, V.E.: Square Rhomboids and Their Algebraic Expressions. In: Proc. 2009 Int. Conf. on Theoretical and Mathematical Foundations of Computer Science (TMFCS-09), pp. 110–117 (2009)

12. Korenblit, M., Levit, V.E.: An Improved Full Decomposition Algorithm for Generating Algebraic Expressions of Square Rhomboids. In: Proc. 2010 Int. Conf. on Theoretical and Mathematical Foundations of Computer Science (TMFCS-10), pp. 31–38 (2010)
13. Mundici, D.: Functions Computed by Monotone Boolean Formulas with no Repeated Variables. Theoretical Computer Science 66, 113–114 (1989)
14. Mundici, D.: Solution of Rota's Problem on the Order of Series-Parallel Networks. Advances in Applied Mathematics 12, 455–463 (1991)
15. Naumann, V.: Measuring the Distance to Series-Parallelity by Path Expressions. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 269–281. Springer, Heidelberg (1995)
16. Rosen, K.H. (ed.): Handbook of Discrete and Combinatorial Mathematics. CRC Press, Boca Raton (2000)
17. Savicky, P., Woods, A.R.: The Number of Boolean Functions Computed by Formulas of a Given Size. Random Structures and Algorithms 13, 349–382 (1998)
18. Wang, A.R.R.: Algorithms for Multilevel Logic Optimization, Ph.D. Thesis, University of California, Berkeley (1989)

Monomial Testing and Applications

Shenshi Chen

Department of Computer Science, University of Texas-Pan American,
Edinburg, TX 78539, USA
schen@broncs.utpa.edu

Abstract. In this paper, we devise two algorithms for the problem of testing q-monomials of degree k in any multivariate polynomial represented by a circuit, regardless of the primality of q. One is an $O^*(2^k)$ time randomized algorithm. The other is an $O^*(12.8^k)$ time deterministic algorithm for the same q-monomial testing problem but requiring the polynomials to be represented by tree-like circuits. Several applications of q-monomial testing are also given, including a deterministic $O^*(12.8^{mk})$ upper bound for the m-set k-packing problem.

Keywords: Group algebra, complexity, multivariate polynomials, monomials, monomial testing randomized algorithms, derandomization.

1 Introduction

Recent research on testing multilinear monomials and q-monomials in multivariate polynomials [13,17,7,8,10,6,9] requires that Z_q be a field, which is true when $q \geq 2$ is prime. When $q > 2$ is not prime, Z_q is no longer a field, hence the group algebra based approaches in [13,17,10,9] become inapplicable. When q is not prime, it remains open whether the problem of testing q-monomials in a multivariate polynomial can be solved in some compatible complexity, such as $O^*(c^k)$ time for a constant $c \geq 2$. Our work in [2] presents a randomized $O^*(7.15^k)$ algorithm for testing q-monomials of degree k in a multivariate polynomial that is represented by a tree-like circuit. This algorithm works for any fixed integer $q \geq 2$, regardless of q's primality. Moreover, for prime $q > 7$, it provides us with some substantial improvement on the time complexity of the previously known algorithm [10,9] for testing q-monomials.

Randomized algebraic techniques have recently led to the once fastest randomized algorithms of time $O^*(2^k)$ for the k-path problem and other problems [13,17]. Another recent seminal example is the improved $O(1.657^n)$ time randomized algorithm for the Hamiltonian path problem by Björklund [3]. This algorithm provided a positive answer to the question of whether the Hamiltonian path problem can be solved in time $O(c^n)$ for some constant $1 < c < 2$, a challenging problem that had been open for half of a century. Björklund *et al.* further extended the above randomized algorithm to the k-path testing problem with $O^*(1.657^k)$ time complexity [4]. Very recently, those two algorithms were simplified further by Abasi and Bshouty [1].

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 106–117, 2013.

This paper consists of three key contributions: The first is an $O^*(2^k)$ time randomized algorithm that gives an affirmative answer to the q-monomial testing problem for polynomials represented by circuits, regardless of the primality of $q \geq 2$. We generalize the circuit reconstruction and variable replacements proposed in [2] to transform the q-monomial testing problem, for polynomials represented by a circuit, into the multilinear monomial testing problem and furthermore enabling the usage of the group algebraic approach originated by Koutis [13] to help resolve the q-monomial testing problem. The second is an $O^*(12.8^k)$ deterministic algorithm for testing q-monomials in multivariate polynomials represented by tree-like circuits. Inspired by the work in [10,9], we devise this deterministic algorithm by derandomizing the first randomized algorithm for tree-like circuits with the help of the perfect hashing functions by Chen *et al.* [11] and the deterministic polynomial identity testing algorithm by Raz and Shpilka [16] for noncommunicative polynomials. The third is to exhibit several applications of q-monomial testing to designing algorithms for concrete problems. Specifically, we show how q-monomial testing can be applied to the non-simple k-path testing problem, the generalized m-set k-packing problem, and the generalized P_2-Packing problem. In particular, we design a deterministic algorithm for solving the m-set k-packing problem in $O^*(12.8^{mk})$, which is, to our best knowledge, the best upper bound for deterministic algorithms to solve this problem.

2 Notations and Definitions

For variables $x_1, \ldots, x_n$, for $1 \leq i_1 < \cdots < i_k \leq n$, $\pi = x_{i_1}^{s_1} \cdots x_{i_t}^{s_t}$ is called a monomial. The degree of π, denoted by $\deg(\pi)$, is $\sum_{j=1}^{t} s_j$. π is multilinear, if $s_1 = \cdots = s_t = 1$, i.e., π is linear in all its variables $x_{i_1}, \ldots, x_{i_t}$. For any given integer $q \geq 2$, π is called a q-monomial if $1 \leq s_1, \ldots, s_t \leq q-1$. In particular, a multilinear monomial is the same as a 2-monomial.

An arithmetic circuit, or circuit for short, is a directed acyclic graph consisting of $+$ gates with unbounded fan-ins, $\times$ gates with two fan-ins, and terminal nodes that correspond to variables. The size, denoted by $s(n)$, of a circuit with n variables is the number of gates in that circuit. A circuit is considered a tree-like circuit if the fan-out of every gate is at most one, i.e., the underlying directed acyclic graph that excludes all the terminal nodes is a tree. In other words, in a tree-like circuit, only the terminal nodes can have more than one fan-out (or out-going edge).

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

By definition, any polynomial $F(x_1, \ldots, x_n)$ can be expressed as a sum of a list of monomials, called the sum-product expansion. The degree of the polynomial is the largest degree of its monomials in the expansion. With this expanded expression, it is trivial to see whether $F(x_1, \ldots, x_n)$ has a multilinear monomial, or a monomial with any given pattern. Unfortunately, such an expanded expression is essentially problematic and infeasible due to the fact that a polynomial

may often have exponentially many monomials in its sum-product expansion. The challenge then is to test whether $F(x_1, \ldots, x_n)$ has a multilinear monomial, or any other desired monomial, efficiently but without expanding it into its sum-product representation.

For any integer $k \geq 1$, we consider the group Z_2^k with the multiplication $\cdot$ defined as follows. For k-dimensional column vectors $\boldsymbol{x}, \boldsymbol{y} \in Z_2^k$ with $\boldsymbol{x} = (x_1, \ldots, x_k)^T$ and $\boldsymbol{y} = (y_1, \ldots, y_k)^T$, $\boldsymbol{x} \cdot \boldsymbol{y} = (x_1 + y_1, \ldots, x_k + y_k)^T$. $\boldsymbol{v}_0 = (0, \ldots, 0)^T$ is the zero element in the group. For any field $\mathcal{F}$, the group algebra $\mathcal{F}[Z_2^k]$ is defined as follows. Every element $u \in \mathcal{F}[Z_2^k]$ is a linear sum of the form

$$u = \sum_{\boldsymbol{x}_i \in Z_2^k,\ a_i \in \mathcal{F}} a_i \boldsymbol{x}_i. \tag{1}$$

For any element $v = \sum_{\boldsymbol{x}_i \in Z_2^k,\ b_i \in \mathcal{F}} b_i \boldsymbol{x}_i$, We define

$$u + v = \sum_{a_i,\ b_i \in \mathcal{F},\ \boldsymbol{x}_i \in Z_2^k} (a_i + b_i)\boldsymbol{x}_i, \text{ and}$$

$$u \cdot v = \sum_{a_i,\ b_j \in \mathcal{F},\ \text{and}\ \boldsymbol{x}_i,\ \boldsymbol{y}_j \in Z_2^k} (a_i b_j)(\boldsymbol{x}_i \cdot \boldsymbol{y}_j).$$

For any scalar $c \in \mathcal{F}$,

$$cu = c \left(\sum_{\boldsymbol{x}_i \in Z_2^k,\ a_i \in \mathcal{F}} a_i \boldsymbol{x}_i \right) = \sum_{\boldsymbol{x}_i \in Z_2^k,\ a_i \in \mathcal{F}} (ca_i)\boldsymbol{x}_i.$$

The zero element in the group algebra $\mathcal{F}[Z_2^k]$ is $\mathbf{0} = \sum_{\boldsymbol{v}} 0\boldsymbol{v}$, where 0 is the zero element in $\mathcal{F}$ and $\boldsymbol{v}$ is any vector in Z_2^k. For example, $\mathbf{0} = 0\boldsymbol{v_0} = 0\boldsymbol{v}_1 + 0\boldsymbol{v}_2 + 0\boldsymbol{v}_3$, for any $\boldsymbol{v}_i \in Z_2^k$, $1 \leq i \leq 3$. The identity element in the group algebra $\mathcal{F}[Z_2^k]$ is $\mathbf{1} = 1\boldsymbol{v}_0 = \boldsymbol{v}_0$, where 1 is the identity element in $\mathcal{F}$. For any vector $\boldsymbol{v} = (v_1, \ldots, v_k)^T \in Z_2^k$, for $i \geq 0$, let $(\boldsymbol{v})^i = (iv_1, \ldots, iv_k)^T$. In particular, when the field $\mathcal{F}$ is Z_2 (or in general, of characteristic 2), in the group algebra $\mathcal{F}[Z_2^k]$, for any $\boldsymbol{z} \in Z_2^k$ we have $(\boldsymbol{v})^0 = (\boldsymbol{v})^2 = \boldsymbol{v}_0$, and $\boldsymbol{z} + \boldsymbol{z} = \mathbf{0}$.

3 A New Transformation

3.1 A New Circuit Reconstruction Method

In this section and the next, we shall extend the transformation methods designed in [2] to general circuits. The circuit reconstruction and variable replacement methods developed by us in [2] work for tree-like circuits only. In essence, the methods are as follows: Replace each original variable x in the polynomial by a + gate g; for each outgoing edge of x, duplicate a copy of g; for each g, allow it to receive inputs from $q - 1$ many new y-variables; for each edge from a y-variable to a duplicated gate g, replace it with a new $\times$ gate that receives

inputs from the y-variable and a new z-variable that then feeds the output to g. Additionally, the methods add a new $\times$ gate f' that multiplies the output of f with a new z-variable for each $\times$ gate f in the original circuit.

For any given polynomial $F(x_1, x_2, \ldots, x_n)$ represented by a circuit $\mathcal{C}$ of size $s(n)$, we first reconstruct the circuit $\mathcal{C}$ in three steps as follows:

Duplicating + Gates. Starting at the bottom layer of the circuit $\mathcal{C}$, for each $+$ gate g with outgoing edges $f_1, f_2, \ldots, f_\ell$, replace g with ℓ copies $g_1, g_2, \ldots, g_\ell$ such that each g_i has the same input as g, but the only outgoing edge of g_i is f_i, $1 \leq i \leq \ell$.

Duplicating Terminal Nodes. For each variable x_i, if x_i is the input to a list of gates $g_1, g_2, \ldots, g_\ell$, then create ℓ terminal nodes $u_1, u_2, \ldots, u_\ell$ such that each of them represents a copy of the variable x_i and g_j receives input from u_j, $1 \leq j \leq \ell$.

Let $\mathcal{C}^*$ denote the reconstructed circuit after the above two reconstruction steps. Obviously, both circuits $\mathcal{C}$ and $\mathcal{C}^*$ compute the same polynomial F.

Adding New $\times$ Gates and New Variables. Having completed the reconstruction to obtain $\mathcal{C}^*$, we then expand it to a new circuit $\mathcal{C}'$ as follows. For every edge e_i in $\mathcal{C}^*$ (including every edge between a gate and a terminal node) such that e_i conveys the output of u_i to v_i, add a new $\times$ gate g_i that multiplies the output of u_i with a new variable z_i and passes the outcome to v_i.

Assume that a list of h new z-variables $z_1, z_2, \ldots, z_h$ have been introduced into the circuit $\mathcal{C}'$. Let $F'(z_1, z_2, \ldots, z_h, x_1, x_2, \ldots, x_n)$ be the new polynomial represented by $\mathcal{C}'$.

Lemma 1. *Let the t be the length of longest path from the root gate of $\mathcal{C}$ to its terminal nodes. $F(x_1, x_2, \ldots, x_n)$ has a monomial π of degree k in its sum-product expansion if and only if there is a monomial $\alpha\pi$ in the sum-product expansion of $F'(z_1, z_2, \ldots, z_h, x_1, x_2, \ldots, x_n)$ such that α is a multilinear monomial of z-variables with degree $\leq tk + 1$. Furthermore, if π occurs more than once in the sum-product expansion of F', then every occurrence of π in F' has a unique coefficient α; and any two different monomials of x-variables in F' will have different coefficients that are multilinear products of z-variables.*

3.2 Variable Replacements

Following Subsection 3.1, we continue to address how to further transform the new polynomial $F'(z_1, z_2, \ldots, z_h, x_1, x_2, \ldots, x_n)$ computed by the circuit $\mathcal{C}'$. The method for this part of the transformation is similar to, but different from, the method proposed by us in [2].

Variable Replacements: Here, we start with the new circuit $\mathcal{C}'$ that computes $F'(z_1, z_2, \ldots, z_h, x_1, x_2, \ldots, x_n)$. For each variable x_i, we replace it with a "weighted" linear sum of $q-1$ new y-variables $y_{i1}, y_{i2}, \ldots, y_{i(q-1)}$. The replacements work as follows: For each variable x_i, we first add $q-1$ new terminal nodes that represent $q-1$ many y-variables $y_{i1}, y_{i2}, \ldots, y_{i(q-1)}$. Then, for each terminal

node u_j representing x_i in $\mathcal{C}'$, we replace u_j with a + gate. Later, for each new + gate g_j that is created for u_j of x_i, let g_j receive input from $y_{i1}, y_{i2}, \ldots, y_{i(q-1)}$. That is, we add an edge from each of such y-variables to g_j. Finally, for each edge e_{ij} from y_{ij} to g_j, replace e_{ij} by a new $\times$ gate that takes inputs from y_{ij} and a new z-variable z_{ij} and sends the output to g_j.

Let $\mathcal{C}''$ be the circuit resulted from the above transformation, and

$$G(z_1, \ldots, z_h, y_{11}, \ldots, y_{1(q-1)}, \ldots, y_{n1}, \ldots, y_{n(q-1)})$$

be the polynomial computed by the circuit $\mathcal{C}''$.

Lemma 2. *Let $F(x_1, x_2, \ldots, x_n)$ be any given polynomial represented by a circuit $\mathcal{C}$ and t be the length of the longest path of $\mathcal{C}$. For any fixed integer $q \geq 2$, F has a q-monomial of x-variables with degree k, then G has a unique multilinear monomial $\alpha\pi$ such that π is a degree k multilinear monomial of y-variables and α is a multilinear monomial of z-variables with degree $\leq k(t+1)+1$. If F has no q-monomials, then G has no multilinear monomials of y-variables, i.e., G has no monomials of the format $\beta\phi$ such that β is a monomial of z-variables and ϕ is a multilinear monomial of y-variables.*

4 A Faster Randomized Algorithm

Consider any given polynomial $F(x_1, x_2, \ldots, x_n)$ that is represented by a circuit $\mathcal{C}$ of size $s(n)$. Note that the length of the longest path from the root of $\mathcal{C}$ to any terminal node is no more than $s(n)$. Let $d = \log_2(k(s(n)+1)+1)+1$ and $\mathcal{F} = \mathrm{GF}(2^d)$ be a finite field of 2^d many elements. We consider the group algebra $\mathcal{F}[Z_2^k]$.

Algorithm RTM (Randomized Testing of q-Monomials):

1. As described in Subsections 3.1 and 3.2, reconstruct the circuit $\mathcal{C}$ to obtain $\mathcal{C}^*$ that computes the same polynomial F and then introduce new z-variables to $\mathcal{C}^*$ to obtain the new circuit $\mathcal{C}'$ that computes $F'(z_1, z_2, \ldots, z_h, x_1, x_2, \ldots, x_n)$. Finally, obtain a circuit $\mathcal{C}''$ by variable replacements so that F' is transformed to

$$G(z_1, \ldots, z_h, y_{11}, \ldots, y_{1(q-1)}, \ldots, y_{n1}, \ldots, y_{n(q-1)}).$$

2. Select uniform random vectors $\boldsymbol{v}_{ij} \in Z_2^k - \{\boldsymbol{v}_0\}$, and replace the variable y_{ij} with $(\boldsymbol{v}_{ij} + \boldsymbol{v}_0)$, $1 \leq i \leq n$ and $1 \leq j \leq q-1$.

3. Use $\mathcal{C}''$ to calculate

$$\begin{aligned} G' &= G(z_1, \ldots, z_h, (\boldsymbol{v}_{11} + \boldsymbol{v}_0), \ldots, (\boldsymbol{v}_{1(q-1)} + \boldsymbol{v}_0), \ldots, \\ &\qquad (\boldsymbol{v}_{n1} + \boldsymbol{v}_0), \ldots, (\boldsymbol{v}_{n(q-1)} + \boldsymbol{v}_0)) \\ &= \sum_{j=1}^{2^k} f_j(z_1, \ldots, z_h) \cdot \boldsymbol{v}_j, \end{aligned} \tag{2}$$

where each f_j is a polynomial of degree $\leq k(s(n)+1)+1$ (see Lemma 2) over the finite field $\mathcal{F} = \mathrm{GF}(2^d)$, and $\boldsymbol{v}_j$ with $1 \leq j \leq 2^k$ are the 2^k distinct vectors in Z_2^k.

4. Perform polynomial identity testing with the Schwartz-Zippel algorithm [14] for every f_j over $\mathcal{F}$. Return *"yes"* if one of those polynomials is not identical to zero. Otherwise, return *"no"*.

It should be pointed out that the actual implementation of Step 4 would be running the Schwartz-Zippel algorithm concurrently for all f_j, $1 \leq j \leq 2^k$, utilizing the circuit $\mathcal{C}''$. If one of those polynomials is not identical to zero, then the output of G' as computed by circuit $\mathcal{C}''$ is not zero.

The group algebra technique established by Koutis [13] assures the following two properties:

Lemma 3. *([13]) Replacing all the variables y_{ij} in G with group algebraic elements $\boldsymbol{v}_{ij} + \boldsymbol{v}_0$ will make all monomials $\alpha\pi$ in G' to become zero, if π is non-multilinear with respect to y-variables. Here, α is a product of z-variables.*

Lemma 4. *([13]) Replacing all the variables y_{ij} in G with group algebraic elements $\boldsymbol{v}_{ij} + \boldsymbol{v}_0$ will make any monomial $\alpha\pi$ to become zero, if and only if the vectors $\boldsymbol{v}_{ij}$ are linearly dependent in the vector space Z_2^k. Here, π is a multilinear monomial of y-variables and α is a product of z-variables, Moreover, when π becomes non-zero after the replacements, it will become the sum of all the vectors in the linear space spanned by those vectors.*

Theorem 1. *Let $q > 2$ be any fixed integer and $F(x_1, x_2, \ldots, x_n)$ be an n-variate polynomial represented by a circuit $\mathcal{C}$ of size $s(n)$. Then, the randomized algorithm RTM can decide whether F has a q-monomial of degree k in its sum-product expansion in time $O^*(2^k s^6(n))$.*

Since we are often interested in circuits with polynomial sizes in n, the time complexity of algorithm RTM is $O^*(2^k)$ for those circuits.

Proof. From the introduction of the new z-variables to the circuit $\mathcal{C}'$, it is easy to see that every monomial in F' has the format $\alpha\pi$, where π is a product of x-variables and α is a product of z-variables. Since only x-variables are replaced by their respective "weighted" linear sums of new y-variables as specified in Subsection 3.2, monomials in G have the format $\beta\phi$, where ϕ is a product of y-variables and β is a product of z-variables.

Suppose that F has no q-monomials. By Lemma 2, G has no monomials $\beta\phi$ such that ϕ is multilinear with respect to y-variables. Moreover, by Lemma 3, replacing y-variables by group algebraic elements at Step 2 will make ϕ in every monomial $\beta\phi$ in G to become zero. Hence, the group algebraic replacements will make G to become zero and so the algorithm RTM will return *"no"*.

Assume that F has a q-monomial of degree k. By Lemma 2, G has a monomial $\beta\phi$ such that ϕ is a multilinear monomial of degree k with respect to y variables and β is a multilinear monomial of degree $\leq k(s(n) + 1) + 1$ with respect to z-variables. It follows from a lemma in [5] (see also, [2]) , that a list of uniform random vectors from Z_2^k will be linearly independent with probability at least 0.28. By Lemma 4, with probability at least 0.28, the multilinear monomial ϕ

will not be annihilated by the group algebraic replacements at Step 2. Precisely, with probability at least 0.28, $\beta\phi$ will become

$$\lambda(\beta\phi) = \sum_{i=1}^{2^k} \beta \boldsymbol{v}_i, \tag{3}$$

where $\boldsymbol{v}_i$ are distinct vectors in Z_2^k.

Let $\mathcal{S}$ be the set of all those multilinear monomials $\beta\phi$ that survive the group algebraic replacements for y-variables in G. Then,

$$\begin{aligned} G' &= G(z_1, \ldots, z_h, (\boldsymbol{v}_{11} + \boldsymbol{v}_0), \ldots, (\boldsymbol{v}_{1(q-1)} + \boldsymbol{v}_0), \ldots, \\ &\qquad (\boldsymbol{v}_{n1} + \boldsymbol{v}_0), \ldots, (\boldsymbol{v}_{n(q-1)} + \boldsymbol{v}_0)) \\ &= \sum_{\beta\phi \in \mathcal{S}} \lambda(\beta\phi) \\ &= \sum_{\beta\phi \in \mathcal{S}} \left(\sum_{i=1}^{2^k} \beta \boldsymbol{v}_i \right) \\ &= \sum_{j=1}^{2^k} \left(\sum_{\beta\phi \in \mathcal{S}} \beta \right) \boldsymbol{v}_j \end{aligned} \tag{4}$$

Let

$$f_j(z_1, \ldots, z_h) = \sum_{\beta\phi \in \mathcal{S}} \beta.$$

By Lemmas 2 and 3, the degree of β is at most $k(s(n) + 1) + 1$. Hence, the coefficient polynomial f_j with respect to $\boldsymbol{v}_j$ in G' after the group algebraic replacements has a degree $\leq k(s(n)+1)+1$. Also, by Lemma 2, β is unique with respect to every ϕ for each monomial $\beta\phi$ in G. Thus, the possibility of a "zero-sum" of coefficients from different surviving monomials is completely avoided during the computation for f_j. Therefore, conditioned on that $\mathcal{S}$ is not empty, G' must not be identical to zero, i.e., there exists at least one f_j that is not identical to zero. At Step 4, we use the randomized algorithm by Schwartz-Zippel [14] to test whether f_j is identical to zero. Since the degree of each f_j is at most $k(s(n) + 1) + 1$, it is known that this testing can be done with probability at least $1 - \frac{\deg(f_j)}{|\mathcal{F}|} \geq \frac{1}{2}$ in time polynomially in $s(n)$ and $\log_2 |\mathcal{F}| = \log_2(k(s(n)+1)+1)+1$. Since $\mathcal{S}$ is not empty with probability at least 0.28, the success probability of testing whether G has a degree k multilinear monomial of y-variables is at least $0.28 \times \frac{1}{2} > \frac{1}{8}$.

Finally, we address the issues of how to calculate G' and the time needed to do so. Naturally, every element in the group algebra $\mathcal{F}[Z_2^k]$ can be represented by a vector in $Z_2^{2^k}$. Adding two elements in $\mathcal{F}[Z_2^k]$ is equivalent to adding the two corresponding vectors in $Z_2^{2^k}$, and the latter can be done in $O(2^k \log_2 |\mathcal{F}|)$ time via

component-wise sum. In addition, multiplying two elements in $\mathcal{F}[Z_2^k]$ is equivalent to multiplying the two corresponding vectors in $Z_2^{2^k}$, and the latter can be done in $O(k2^{k+1}\log_2|\mathcal{F}|)$ with the help of a similar Fast Fourier Transform style algorithm as in Williams [17]. By the circuit reconstruction and variable replacements in Subsections 3.1 and 3.2, the size of the circuit $\mathcal{C}''$ is at most $s^3(n)$. Calculating G' by the circuit $\mathcal{C}''$ consists of $n * s^6(n)$ arithmetic operations of either adding or multiplying two elements in $\mathcal{F}[Z_2^k]$ based on the circuit $\mathcal{C}''$. Hence, the total time needed is $O(n * s^6(n)k2^{k+1}\log_2|\mathcal{F}|)$. At Step 4, we run the Schwartz-Zippel algorithm on G' to simultaneously test whether there is one f_j such that f_j is not identical to zero. Recall that $\log_2|\mathcal{F}| = log_2(k(s(n)+1)+1)+1$. The total time for the entire algorithm is $O^*(2^k s^6(n))$.

5 A Deterministic Algorithm via Derandomization

Definition 1. *(See, Chen et al. [11]) Let n and k be two integers such that $1 \leq k \leq n$. Let $\mathcal{A} = \{1, 2, \ldots, n\}$ and $\mathcal{K} = \{1, 2, \ldots, k\}$. A k-coloring of the set $\mathcal{A}$ is a function from $\mathcal{A}$ to $\mathcal{K}$. A collection $\mathcal{F}$ of k-colorings of $\mathcal{A}$ is a (n, k)-family of* perfect hashing functions *if for any subset W of k elements in $\mathcal{A}$, there is a k-coloring $h \in \mathcal{F}$ that is injective from W to $\mathcal{K}$, i.e., for any $x, y \in W$, $h(x)$ and $h(y)$ are distinct elements in $\mathcal{K}$.*

Theorem 2. *Let $q \geq 2$ be fixed integer. Let $F(x_1, x_2, \ldots, x_n)$ be an n-variate polynomial of degree k represented by a tree-like circuit $\mathcal{C}$ of size $s(n)$. There is a deterministic $O^*(12.8^k s^6(n))$ time algorithm to test whether F has a q-monomial of degree k in its sum-product expansion.*

Proof. Let $d = \log_2(k(s(n) + 1) + 1) + 1$ and $\mathcal{F} = \mathrm{GF}(2^d)$ be a finite field of 2^d elements. The deterministic algorithm DTM for testing whether F has a q-monomial of degree k is given as follows.

Algorithm DTM (Deterministic Testing of q-Monomials):

1. As in the Algorithm RTM, following circuit reconstruction and variable replacements in Subsections 3.1 and 3.2, reconstruct the circuit $\mathcal{C}$ to obtain $\mathcal{C}^*$ that computes the same polynomial F and then introduce new z-variables to $\mathcal{C}^*$ to obtain the new circuit $\mathcal{C}'$ that computes $F'(z_1, z_2, \ldots, z_h, x_1, x_2, \ldots, x_n)$. Finally, perform variable replacements to obtain the circuit $\mathcal{C}''$ that transforms F' to

$$G(z_1, \ldots, z_h, y_{11}, \ldots, y_{1(q-1)}, \ldots, y_{n1}, \ldots, y_{n(q-1)}).$$

2. Construct with the algorithm by Chen *at el.* [11] a $((q-1)ns(n), k)$-family of perfect hashing functions $\mathcal{H}$ of size $O(6.4^k \log_2^2((q-1)ns(n)))$

3. Select k linearly independent vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in Z_2^k$. (No randomization is needed at this step, either.)

4 For each perfect hashing function $\lambda \in \mathcal{H}$ do

4.1. Let $\gamma(i,j)$ be any given one-to-one mapping from $\{(i,j)|1 \le i \le n \text{ and } 1 \le j \le q-1\}$ to $\{1,2,\ldots,(q-1)n\}$ to label variables y_{ij}. Replace each variable y_{ij} in G with $(\boldsymbol{v}_{\lambda(\gamma(i,j))} + \boldsymbol{v}_0)$, $1 \le i \le n$ and $1 \le j \le q-1$.

4.2. Use $\mathcal{C}''$ to calculate

$$\begin{aligned} G' &= G(z_1,\ldots,z_h,(\boldsymbol{v}_{\lambda(\gamma(1,1))}+\boldsymbol{v}_0),\ldots,(\boldsymbol{v}_{\lambda(\gamma(1,(q-1)))}+\boldsymbol{v}_0),\\ &\qquad \ldots,(\boldsymbol{v}_{\lambda(\gamma(n,1))}+\boldsymbol{v}_0),\ldots,(\boldsymbol{v}_{\lambda(\gamma(n,(q-1)))}+\boldsymbol{v}_0))\\ &= \sum_{j=1}^{2^k} f_j(z_1,\ldots,z_h)\cdot \boldsymbol{v}_j, \end{aligned} \tag{5}$$

where each f_j is a polynomial of degree $\le k(s(n)+1)+1$ (see, Lemma 2) over the finite field $\mathcal{F} = \mathrm{GF}(2^d)$, and $\boldsymbol{v}_j$ with $1 \le j \le 2^k$ are the 2^k distinct vectors in Z_2^k.

4.3. Perform polynomial identity testing with the Raz and Shpilka algorithm [16] for every f_j over $\mathcal{F}$. Stop and return *"yes"* if one of them is not identical to zero.

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

The correctness of algorithm DTM is guaranteed by the nature of perfect hashing and the correctness of algorithm RTM. We shall now focus on analyzing the time complexity of the algorithm.

Note that q is a fixed constant. By Chen *at el.*[11], Step 2 can be done in $O(6.4^k n \log^2((q-1)n)) = O^*(6.4^k)$ time. Step 3 can be easily done in $O(k^2)$ time.

It follows from Lemma 3 that all those monomials that are not q-monomials in F, and hence in F', will be annihilated when variables y_{ij} are replaced by $(\boldsymbol{v}_{\lambda(t(i,j))} + \boldsymbol{v}_0)$ in G at Step 4.1.

Consider any given q-monomial $\pi = x_{i_1}^{s_1}\cdots x_{i_t}^{s_t}$ of degree k in F with $1 \le s_j \le q-1$ and $k = \deg(\pi)$, $j = 1,\ldots,t$. By Lemma 2, there are monomials $\alpha\pi$ in F' such that α is a multilinear monomial of z-variables with degree $\le k(s(n)+1)+1$, and all such monomials are distinct. By Lemma 4, π (hence, $\alpha\pi$) will survive the replacements at Step 4.1. Let $\mathcal{S}$ be the set of all the surviving q-monomials $\alpha\pi$. Following the same analysis as in the proof of Theorem 1, we have

$$\begin{aligned} G' &= G(z_1,\ldots,z_h,(\boldsymbol{v}_{\lambda(\gamma(1,1))}+\boldsymbol{v}_0),\ldots,(\boldsymbol{v}_{\lambda(\gamma(1,(q-1)))}+\boldsymbol{v}_0),\\ &\qquad \ldots,(\boldsymbol{v}_{\lambda(\gamma(n,1))}+\boldsymbol{v}_0),\ldots,(\boldsymbol{v}_{\lambda(\gamma(n,(q-1)))}+\boldsymbol{v}_0))\\ &= \sum_{j=1}^{2^k}\left(\sum_{\beta\phi\in\mathcal{S}}\beta\right)\boldsymbol{v}_j\\ &= \sum_{j=1}^{2^k} f_j(z_1,\ldots,z_h)\boldsymbol{v}_j\\ &\neq 0 \end{aligned}$$

since $\mathcal{S}$ is not empty. Here,

$$f_j(z_1, \ldots, z_h) = \sum_{\beta\phi \in \mathcal{S}} \beta.$$

This means that, conditioned on that $\mathcal{S}$ is not empty, there is at least one f_j that is not identical to zero. Again, as in the analysis for algorithm RTM, the time needed for calculating G' is $O^*(2^k s^6(n))$ when the replacements are fixed for x-variables and the subsequent algebraic replacements are given for y-variables.

We now consider imposing noncommunicativity on z-variables in $\mathcal{C}''$. This can be done by imposing an order for z-variable inputs to any gates in $\mathcal{C}''$. Technically, however, we shall allow values for z-variables to communicate with those for y-variables. Finally, we use the algorithm by Raz and Shpilka [16] to test whether $f_j(z_1, \ldots, z_h)$ is identical to zero of not. This can be done in time polynomially in $s(n)$ and n, since with the imposed order for z-variables f_j is a non-communicative polynomial represented by a tree-like circuit.

Combining the above analysis, the total time of the algorithm DTM is $O^*(6.4^k \times 2^k s^6(n)) = O^*(12.8^k s^6(n))$.

When the circuit size $s(n)$ is a polynomial in n, the time bound becomes $O^*(12.8^k)$.

6 Applications

We list three applications of the q-monomial testing to concrete algorithm designs. Here, we assume $q \geq 2$ is a fixed integer. Notably, algorithm DTM can help us to derive a deterministic algorithm for solving the m-set k-packing problem in $O^*(12.8^{mk})$, which is, to our best knowledge, the best upper bound for deterministic algorithms to solve this problem.

6.1 Allowing Overlapping in m-Set k-Packing

Let $\mathcal{S}$ be a collection of sets so that each member in $\mathcal{S}$ is a subset of an n-element set X. Additional, members in $\mathcal{S}$ have the same size $m \geq 3$. We may like to ask whether there are k members in $\mathcal{S}$ such that those members are either pairwise disjoint or at most $q-1$ members may overlap. This problem with respect to q is a generalized version of the m-Set k-packing problem.

We can view each element in X as a variable. Thus, a member in $\mathcal{S}$ is a monomial of m variables. Let

$$F(\mathcal{S}, k) = \left(\sum_{A \in \mathcal{S}} f(A) \right)^k,$$

where $f(A)$ denotes the monomial derived from A. Then, the above generalized problem m-set k-packing with respect to q is equivalent to ask whether $F(\mathcal{S}, k)$ has a q-monomial of degree mk. Again, algorithm RTM solves this problem in $O^*(2^{mk})$ time. When $q = 2$, the $O^*(2^{mk})$ bound was obtained in [13].

Since $F(\mathcal{S}, k)$ can be represented by a tree-like circuit, we can choose $q = 2$ and apply algorithm DTM to test whether $F(\mathcal{S}, k)$ has multilinear monomial (i.e., 2-monomial) of degree mk. Therefore, we have a deterministic algorithm to solve the m-set k-packing problem in $O^*(12.8^{mk})$ time. Although there are many faster randomized algorithms for solving this problem, for deterministic algorithms our $O^*(12.8^{mk})$ upper bound significantly improves the best known upper bound $O^*(\exp(O(mk)))$ by Fellow *et al.* [12]. The upper bound in [12] has a large hidden constant in the exponent, e.g., in the case of $r = 3$, their upper bound is $O^*((12.7D)^{3m})$ for some $D \geq 10.4$.

6.2 Testing Non-Simple k-Paths

Given any undirected graph $G = (V, E)$ with $|V| = n$, we may like to know whether there is a k-path in G such that the path may have loops but any vertex in the path can appear at most $q-1$ times. It is easy to see that this non-simple k-path problem with respect to q is a generalized version of the simple k-path problem.

For each vertex $v_i \in V$, define a polynomial $F_{k,i}$ as follows:

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

We define a polynomial for G as

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

Obviously, $F(G, k)$ can be represented by an arithmetic circuit. It is easy to see that the graph G has a non-simple k-path with respect to q, if and only if $F(G, k)$ has a q-monomial of degree k. Algorithm RTM can solve this problem in $O^*(2^k)$ time. When $q = 2$, the $O^*(2^k)$ bound was obtained in [13,17].

6.3 A Generalized P_2-Packing Problem

Given any undirected graph $G = (V, E)$ with $|V| = n$ and an integer k, we can collect P_2's from G, i.e., simple paths of length 2 in G. The generalized P_2-packing problem with respect to q asks whether there is a collection of k many P_2's such that either all those P_2's are pairwise disjoint, or at most $q-1$ of them may share a common vertex. The generalized P_2-packing problem with respect to q can be easily transformed to a generalized 3-Set k-Packing problem with respect to q. Thereby, an $O^*(2^{3k})$ time randomized solution is given by algorithm RTM.

Acknowledgment. Shenshi is supported by Dr. Bin Fu's NSF CAREER Award, 2009 April 1 to 2014 March 31. The full version of the present paper is available at http://arxiv.org/abs/1303.0478

References

1. Abasi, H., Bshouty, N.: A simple algorithm for undirected hamiltonicity, ECCC, Report No. 12 (2013)
2. Chen, S., Chen, Y., Yang, Q.: Towards randomized testing of q-monomials in multivariate polynomials (January 2013) (manuscript), `http://arxiv.org/abs/1302.5898`
3. Björklund, A.: Determinant sums for undirected hamilonicity. In: Proceedings of the 51th IEEE Foundations of Computer Science (FOCS 2010), pp. 173–182 (2010)
4. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Narrow sieves for parameterized paths and packings. arXiv:1007.1161v1 (2010)
5. Blum, M., Kannan, S.: Designing programs that check their work. J. ACM 42(1), 269–291 (1995)
6. Chen, Z., Fu, B.: Approximating Multilinear Monomial Coefficients and Maximum Multilinear Monomials in Multivariate Polynomials. J. Comb. Optim. 25(2), 234–254 (2013)
7. Chen, Z., Fu, B.: The Complexity of Testing Monomials in Multivariate Polynomials. In: Wang, W., Zhu, X., Du, D.-Z. (eds.) COCOA 2011. LNCS, vol. 6831, pp. 1–15. Springer, Heidelberg (2011)
8. Chen, Z., Fu, B.: Approximating Multilinear Monomial Coefficients and Maximum Multilinear Monomials in Multivariate Polynomials. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part I. LNCS, vol. 6508, pp. 309–323. Springer, Heidelberg (2010)
9. Chen, Z., Fu, B., Liu, Y., Schweller, R.T.: On Testing Monomials in Multivariate Polynomials. Theoretical Computer Science (April 13, 2012) (forthcoming), doi:10.1016/j.tcs.2012.03.038
10. Chen, Z., Fu, B., Liu, Y., Schweller, R.: Algorithms for Testing Monomials in Multivariate Polynomials. In: Wang, W., Zhu, X., Du, D.-Z. (eds.) COCOA 2011. LNCS, vol. 6831, pp. 16–30. Springer, Heidelberg (2011)
11. Chen, J., Lu, S., Sze, S.-H., Zhang, F.: Improved algorithms for path, matching, and packing problems. In: SODA, pp. 298–307 (2007)
12. Fellows, M.R., Knauer, C., Nishimura, N., Ragde, P., Rosamond, F., Stege, U., Thilikos, D.M., Whitesides, S.: Faster fixed-parameter tractable algorithms for matching and packing problems. Algorithmica 52(2), 167–176 (2008)
13. Koutis, I.: Faster algebraic algorithms for path and packing problems. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 575–586. Springer, Heidelberg (2008)
14. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
15. Naor, M., Schulman, L.J., Srinivasan, A.: Splitters and near-optimal derandomization. In: FOCS, pp. 182–191 (1995)
16. Raz, R., Shpilka, A.: Deterministic polynomial identity testing in non-commutative models. Computational Complexity 14(1), 1–19 (2005)
17. Williams, R.: Finding paths of length k in $O^*(2^k)$ time. Information Processing Letters 109, 315–318 (2009)

The Optimal Rescue Path Set Problem in Undirected Graphs

Huili Zhang[1,2,⋆] and Yinfeng Xu[1,2]

[1] School of Management, Xi'an Jiaotong University, Xi'an, 710049, China
[2] State Key Lab for Manufacturing Systems Engineering, Xi'an, 710049, China
zhang.huili@stu.xjtu.edu.cn, yfxu@mail.xjtu.edu.cn

Abstract. This paper proposes the *optimal rescue path set problem* in an undirected graph $G = (V, E)$, in which some vehicles have to go from a source node s to a destination node t. However, during the traveling of the vehicles, there might exist one edge blocked in the graph. The goal is to find a minimum collection of paths for the vehicles, to guarantee the fastest arrival of at least one vehicle no matter which edge is blocked. We present an algorithm for the above optimal rescue path set problem, and prove that the complexity of our algorithm is $O(m + n \log n)$.

Keywords: shortest path, joint replacement path, optimal rescue path set, multiple vehicles.

1 Introduction

Rescue is critical for the management of disasters, such as fire, traffic accidents and so on. Obviously, the rescue vehicles should arrive at the disaster site as soon as possible. However, the shortest path in the graph may not ensure the fastest arrival because of the uncertain traffic congestion caused by the disaster. Moreover, it is hard to forecast when and where the congestion will occur under the emergency situation. Due to this, there are abundant researches on the problem of how to select a path or a path set to make the response as fast as possible.

One obvious solution is to find multiple paths for more than one vehicles. The multiple paths will share the risk caused by the uncertain road blockage. One example is the *k shortest paths problem*. Yen[1] showed how to compute the k-simple shortest paths in weighted directed graphs. The running time of their algorithms, when modern data structures are used, is $O(k(mn + n^2 \log n))$. Eppstein [2] presented an $O(m + n \log n + k)$-time algorithm for the directed version of this problem, when the paths are not required to be simple. For the restricted case of undirected graphs, Katoh et al. [3] presented an algorithm with a running time of O(k(m + n log n)), when modern data structures are used. Unfortunately, the algorithm of k shortest paths often provides very similar paths between s and t, which means that the paths overlap in some segments. If the overlapping edges are blocked, the multiple paths still can't effectively

⋆ Corresponding author.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 118–129, 2013.

decrease the response delay caused by uncertain blockage. Therefore, Akgun et al. [4] presented the *k dissimilar paths problem* by defining certain dissimilar criterion. Some variants of this problem are also studied [5,6].

The other solution is to find an alternative path for one vehicle. The related research includes two aspects: the *replacement path problem* and the *longest detour path problem*.

The *replacement path* is the shortest path between source s and destination t when some edge e_i is removed. Malik et al. [7] presented an $O(m + n \log n)$-time algorithm to find the most vital edge whose replacement path is longest. An $O(m \cdot \alpha(m, n))$-time algorithm was given for the problem by Nardelli et al. [8]. These results apply for undirected graphs only. Emek et al. [9] presented a near-linear time algorithm (in $O(n(\log^3 n)$ time) for computing replacement paths in weighted planar directed graphs. Wulff-Nilsen [10] improved the result to $O(n(\log n)$ time.

The *longest detour path* was introduced by Nardelli et al. [11]. The detour path is defined as the shortest path from v_i to t with removal of the edge $e(v_i, v_j)$. Nardelli et al. [11] redefined the vital edge of a shortest path as that with regards to the longest detour path and gave an algorithm with the same time and space complexity as Malik et al. does [7]. Nardelli et al. [8] also improved the result to $O(m \cdot \alpha(m, n))$. Based on the concept of detour path, Xiao et al. [12] proposed the concept of real time detour path which is introduced in Section 2. Furthermore, the authors proposed the *anti-risk path problem* which aims to find a path P with minimum risk, where the risk is the maximum of the lengths of path P and the realtime detour paths with regards to edges on P .

The previous two solutions try to reduce the risk of blockage by choosing more vehicles to share the risk or by finding a risk-averse route for one vehicle. However, these solutions can not ensure the fastest response to the emergency because the blocked edge is uncertain in advance.

We contribute to the current research by raising the question of how many vehicles are necessary to make sure the fastest response when there is at most one edge blocked and the blockage information is unavailable in advance. The path set for these vehicles is called the *optimal rescue path set*. This path set contains the fewest paths, but achieves the goal of fastest response, i.e. the vehicle arriving first chooses a shortest path on the current graph excluding the blocked edge if there is one. Furthermore, we give an algorithm *Subtrees Connected* to compute the optimal rescue path set and prove its time complexity.

The rest of this paper is as follows. Section 2 lists a collection of assumptions and defines some basic concepts, including the optimal rescue path set. Section 3 analyzes the properties of the optimal rescue path set, and introduces the concept of joint replacement path as well the *Least Overlap* algorithm for it. In section 4, we design the *Subtrees Connected* algorithm to compute the optimal rescue path set and also analyze the time complexity of the *SC* algorithm. Section 5 summarizes the main results of the paper.

2 Preliminaries

We consider a 2-connected undirected graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = m$ edges, where each edge $e(i, j) = (v_i, v_j) \in E$ is associated with a traverse time $w(i, j)$. For two prescribed nodes s and t in V, which are the source node and the destination node respectively, we use $\Omega(s, t)$ to denote the set of all the simple paths (i.e., no node in the path is visited more than once) from s to t. For any path $P(s, t)$ from s to t, let $dP(s, t)$ denote the length of $P(s, t)$. In addition, we use $SP_G(s, t)$ to denote a shortest path from s to t in graph G. We first introduce some basic definitions.

Definition 1. Real time detour path *[12]. When an edge $e(i, j)$ on path $P(s, t)$ is blocked, let $SP_{G-\{e(i,j)\}}(v_i, t)$ be a shortest path from v_i to t on graph $G - \{e(i, j)\}$, then the path $P(e(i, j)) = P(s, v_i) + SP_{G-\{e(i,j)\}}(v_i, t)$ is called a real time detour path of $P(s, t)$ with regards to $e(i, j)$.*

Definition 2. Replacement path *[13]. When an edge $e(i, j)$ on path $P(s, t)$ is blocked, any shortest path $SP_{G-\{e(i,j)\}}(s, t)$ in graph $G - \{e(i, j)\}$ is called a replacement path with regards to $e(i, j)$.*

Notice that for an edge $e(i, j)$ on path $P(s, t)$, there may exist multiple real time detour paths (and multiple replacement paths) of $P(s, t)$ with regards to $e(i, j)$. See Fig. 1 as an example of a replacement path and a real time detour path.

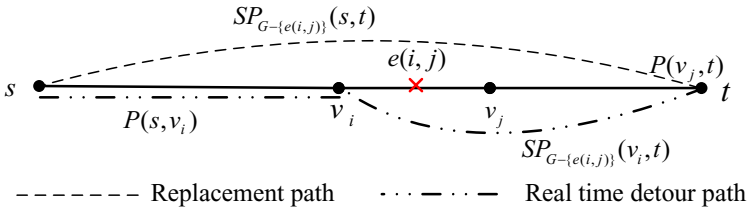


Fig. 1. Replacement path and real time detour path

Given a 2-connected undirected graph G, and two specified nodes s and t in G as the source node and the destination node, respectively. A collection $\mathtt{C}$ of paths (each connects s and t) in G is called a *rescue path set* if: (1) at least one path in $\mathtt{C}$ is a shortest path from s to t in G; (2) for every edge $e(i, j)$ in G, either some path $P \in \mathtt{C}$ is a shortest path from s to t in $G - \{e(i, j)\}$, or some path $P \in \mathtt{C}$ satisfies that $e(i, j) \in P$ and the real time detour path of P with regards to $e(i, j)$ is a shortest path from s to t in $G - \{e(i, j)\}$. An ***optimal rescue path set***(denote as $\mathtt{C}^*$) is a rescue path set containing the minimum number of paths.

Our discussion is based on the following assumptions:

(1) There is at most one edge in G is blocked.

(2) There is one unique shortest path $SP_G(s, t)$ from s to t in G.

Note: Without loss of generality, let $SP_G(s, t)$ be $s = v_1, v_2, \cdots, v_p = t$ and $E_{sp} = \{e(i, i+1) \,|i = 1, \cdots, p-1\}$ be set of edges on $SP_G(s, t)$. Denote $e(i, i+1) \in E_{sp}$ as e_i for short. Define $\{e_i \,|i = 1, \cdots, j-1, e_i \in E_{sp}\}$ as $\bar{E}_{i,j}$.

3 Properties of Optimal Rescue Path Set

According to the above definition of an optimal rescue path set, it has the following two properties:

(1) fastest response, i.e. it ensures the fastest arrival from s to t no matter which edge in G (or none) is blocked;

(2) minimum cost, i.e. the minimized number of paths in the set (that is, the number of vehicles required) to ensure property(1).

Under uniqueness assumption of the shortest path in G, and by the above definitions of the replacement path and the real time detour path, it is easy to see that a collection $\mathtt{C}$ of paths (each connects s and t) in G is a rescue path set if: (1) $SP_G(s,t) \in \mathtt{C}$; (2) for every edge $e(i,j) \in SP_G(s,t)$, either $\mathtt{C}$ contains a replacement path of $SP_G(s,t)$ with regards to $e(i,j)$, or the length of the real time detour path of $SP_G(s,t)$ with regards to $e(i,j)$ is equal to the length of the replacement path with regards to $e(i,j)$.

Note that the uniqueness assumption is also used in the above second requirement, which implies that for an edge $e(i,j) \in SP_G(s,t)$, the length of the real time detour path of $SP_G(s,t)$ with regards to $e(i,j)$ cannot be larger than that of another path P (where $e(i,j) \in P$) with regards to $e(i,j)$.

It is easy to prove that $\mathtt{C} = \{SP_{G-\{e_i\}}(s,t)\,|i = 1, \cdots, p-1\} \cup \{SP_G(s,t)\}$ is a rescue path set. However, it may not be the one with minimum cost because there are some redundant replacement paths. The redundancies come from the following two aspects:

1) the replacement path and the corresponding realtime detour path are of the same length. When some edge is blocked, the vehicle taking $SP_G(s,t)$ detours, and its track is also the corresponding replacement path. Hence, this replacement path is not necessary.

2) the joint replacement path. Several replacement paths respectively with regards to different edges are of the same length, and these paths can be replaced by a *joint replacement path* of these edges, as the the following subsection introduced.

3.1 Minimum Joint Replacement Path Set

In order to find an optimal rescue path set $\mathtt{C}^*$, we introduce the definition of joint replacement path in advance.

Definition 3. Joint Replacement Path. *For any $e_i \in E_{sp}^{'}$, where $E_{sp}^{'} \subseteq E_{sp}$, $dSP_{G-E_{sp}^{'}}(s,t)) = dSP_{G-\{e_i\}}(s,t)$ holds. $SP_{G-E_{sp}^{'}}(s,t)$ is called the joint replacement path of edges in $E_{sp}^{'}$.*

Note: If there is only one edge in $E_{sp}^{'}$, the joint replacement path is actually the replacement path.

The following is the property of the joint replacement path.

Lemma 1. *$\forall e_i, e_j \in E_{sp}$, if e_i and e_j have the joint replacement path, the path includes none of the edges in $\{e_x | i < x < j\}$.*

Proof. Assume that there is a joint replacement path of e_i and e_j in the graph in Fig. 2 and the path contains an edge $e_x(i < x < j)$, $SP_{G-\{e_i,e_j\}}(s,t) = P_1 + e_x + P_2$. We will derive a contradiction.

According to definition of joint replacement path, $dSP_{G-\{e_i,e_j\}}(s,t) = dSP_{G-\{e_i\}}(s,t)$. Define $P_3 = P_1 + SP_G(v_x,t)$ and $P_4 = e_x + P_2$ be two assistant paths. Because of the assumption 2), i.e. only one shortest paths on the graph, it is holden that $dSP_G(v_x,t) < dP_4$. Furthermore, we have $dP_3 < dSP_{G-\{e_i,e_j\}}(s,t))$. It means that there exists a path P_3 in $G - \{e_i\}$ is shorter than the joint replacement path, which contradicts its definition. Therefore, the assumption doesn't hold. The joint replacement path of e_i and e_j includes none of the edges in $\{e_x | i < x < j\}$. The lemma follows. □

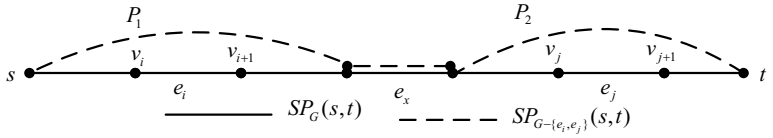


Fig. 2. Joint replacement path

After excluding the edges whose realtime detour paths are of the same length with the corresponding replacement paths, E_{sp} can be divided into different subsets according to lengths of the replacement paths. Define E_{ed} as a subset of E_{sp}. The replacement paths with regards to every edge in E_{ed} are of the same length, denote as D.

Note: To compute the optimal rescue path set $\mathtt{C}^*$, for every E_{ed}, we should select minimum collection of joint replacement paths with regards to the edges in E_{ed}, in which there is at least one replacement path with regards to every edge in E_{ed}. Denote the minimum joint replacement path set with regards to E_{ed} as $\mathtt{C}^*_D$.

Definition 4. *Maximal hunch path(MHP, for short). If* $SP_{G-\bar{E}_{i,j}}(s,t)$ *is* $SP_{G-\bar{E}_{i,j}}(s,t) = SP_G(s,v_i)+SP_{G-\bar{E}_{i,j}}(v_i,v_j)+SP_G(v_j,t)$, *denote* $SP_{G-\bar{E}_{i,j}}(s,t)$ *as* $P_{i,j}$. *If* $dP_{i,j} < dP_{i,j+1}$ *and* $dP_{i,j} < dP_{i-1,j}$, $P_{i,j}$ *is called a maximal hunch path.*

Suppose there are in total X MHPs of the length D, denote $\mathtt{C}_D$ as the collection of the X MHPs corresponding to E_{ed}.

Note: To find $\mathtt{C}^*_D$ with regards to E_{ed} can be viewed as to find a minimum subset of $\mathtt{C}_D$ to ensure that for every edge in E_{ed} there is at least one MHP avoiding this edge.

Definition 5. *Tight replaced edge set(TRES, for short). Define the set* $E^+ = \bar{E}_{i,j} \cap E_{ed}$. *Rename* E^+ *as* $E_{x_{ij},y_{ij}}$, *where* $x_{ij} = \min_{e_g \in E^+}\{g\}$ *and* $y_{ij} = \max_{e_g \in E^+}\{g\}$. $E_{x_{ij},y_{ij}}$ *is called the tight replaced edge set of* $P_{i,j}$.

Let E_{TR} denote the set of TRESs $E_{x_{ij},y_{ij}}$ corresponding MHP $P_{i,j}$ in $\mathtt{C}_D$.

Note: To find a minimum subset of $\mathtt{C}_D$ can be viewed as to find minimum sub set family $\bar{E}^*$ of E_{ed} from E_{TR} satisfying the following condition: $\bigcup_{E_{x_{ij},y_{ij}} \in \bar{E}^*} E_{x_{ij},y_{ij}} = E_{ed}$.

In the following, we will design the Least Overlap algorithm to compute the minimum sub set family $\bar{E}^*$ of E_{ed}.

Least Overlap (LO) Algorithm:

Step 1: Modify E_{TR} to satisfy the condition: $\forall E_{x_{ij},y_{ij}}, E_{x_{cd},y_{cd}} \in \bar{E}$, $x_{ij} < x_{cd}$ and $y_{ij} < y_{cd}$. Sort $E_{x_{ij},y_{ij}} \in E_{TR}$ as the increasing order of x_{ij}(it is also the increasing order of y_{ij}). Let E_{pr} be an order-preserving map from E_{ed} onto $E_{pr} = \{e_g^{'} | g = 1, \cdots, |E_{ed}|\}$, $E_{TR}^{'}$ be an order-preserving map from E_{TR} onto $E_{TR}^{'} = \{E_{x_f,y_f} | f = 1, \cdots, |E_{TR}|\}$, where $x_f < x_{f+1}$ and $y_f < y_{f+1}$. If $E_{x_f,y_f} \in E_{TR}^{'}$ is the map of $E_{x_{ij},y_{ij}} \in E_{TR}$, $e_g^{'}, e_h^{'} \in E_{pr}$ are the map of $e_{x_{ij}}, e_{y_{ij}} \in E_{ed}$ respectively, let $x_f = g$ and $y_f = h$.

Step 2: $\overline{E}^* = \{E_{x_1,y_1}\}$, $a = x_1$, $b = y_1 + 1$;

Step 3: If there exists E_{x_i,y_i} satisfying the condition that $a < x_i \leq b$, let $x_c = \max_{a<x_i \leq b} \{x_i\}$, $\overline{E}^* = \overline{E}^* \cup \{E_{x_c,y_c}\}$, $a = x_c$, $b = y_c$, go to Step 3; otherwise, go to Step 4;

Step 4: Output $\overline{E}^*$, stop.

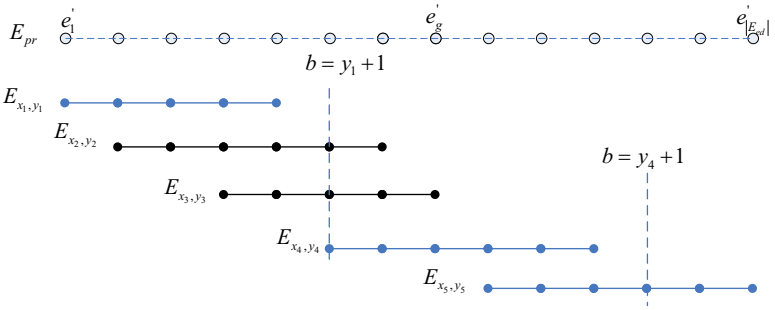


Fig. 3. Least Overlap Algorithm

Example: Let $E_{ed} = \{e_3, e_5, e_8, e_9\}$, and $E_{TR} = \{E_{x_{15},y_{15}}, E_{x_{69},y_{69}}, E_{x_{26},y_{26}}, E_{x_{48},y_{48}}, \}$, where $E_{x_{15},y_{15}} = \{e_3, e_5\}(x_{15} = 3, y_{15} = 5)$, $E_{x_{69},y_{69}} = \{e_8, e_9\}(x_{69} = 8, y_{69} = 9)$, $E_{x_{26},y_{26}} = \{e_3, e_5\}(x_{26} = 3, y_{26} = 5)$, and $E_{x_{48},y_{48}} = \{e_5, e_8\}(x_{48} = 5, y_{48} = 8)$.

In Step 1, since $x_{15} = x_{26}$ and $y_{15} = y_{26}$, $E_{TR} = E_{TR} - E_{x_{26},y_{26}}$. After sorting $\bar{E}$ as the increasing order of $x_{i,j}$, $E_{TR} = \{E_{x_{15},y_{15}} E_{x_{48},y_{48}}, E_{x_{69},y_{69}}\}$. The map of E_{ed} is $\{e_1^{'}, e_2^{'}, e_3^{'}, e_4^{'}\}$. The map of E_{TR} is $E_{TR}^{'} = \{E_{x_1,y_1}, E_{x_2,y_2}, E_{x_3,y_3}\}$, where $x_1 = 1, y_1 = 2, x_2 = 2, y_2 = 3, x_3 = 3, y_3 = 4$.

In step 2, $\overline{E}^* = \{E_{x_1,y_1}\}$, $a = x_1 = 1$, $b = y_1 + 1 = 3$;

In Step 3, $x_2 < x_3 \leq b$, $x_c = x_3$, $a = x_3$, $b = y_3 + 1$, $\overline{E}^* = \{E_{x_1,y_1}, E_{x_3,y_3}\}$. Because no $x_i \leq b$, Stop.

Hence, $\overline{E}^* = \{E_{x_1,y_1}, E_{x_3,y_3}\}$ is the minimum sub set family of E_{ed}, and the $\{P_{1,5}, P_{6,9}\}$ is minimum joint replacement path set with regards to E_{ed}.

The validity of the LO algorithm is checked as follows.

Lemma 2. *Least Overlap Algorithm computes the correct minimum subset family* $\overline{E}^*$ *of* E_{ed}.

Proof. In Step1, $\forall E_{x_{ij},y_{ij}}, E_{x_{cd},y_{cd}} \in E_{TR}$, if $x_{ij} \leq x_{cd}$ and $y_{cd} \leq y_{ij}$, $P_{i,j}$ is not only the replacement path of $E_{x_{ij},y_{ij}}$ but also that of $E_{x_{cd},y_{cd}}$. Therefore, $P_{c,d}$ wouldn't belong to the minimum joint replacement path set, $\overline{E}^*$ derived from $E_{TR} - \{E_{x_{cd},y_{cd}}\}$ will be no worse than that derived from E_{TR}.

Assume that the subset family $\overline{E_{LO}}^* = \overline{E}^* = \{E_{X_i,Y_i} | i = 1, \cdots, n_1\}$ derived by Step 2-4 of LO Algorithm is not the minimum one. We will derive a contradiction.

Suppose that there exists a set $\overline{E_c}^* = \{E_{\overline{X_i},\overline{Y_i}} | i = 1, \cdots, n_2 < n_1\}$ satisfying $\bigcup_{E_{\overline{X_i},\overline{Y_i}} \in \overline{E_c}^*} E_{x_i,y_i} = E_{pr}$. Because E_{x_1,y_1} has to be chosen to cover $e_1^{'}$, $Y_1 = \overline{Y_1} = y_1$.

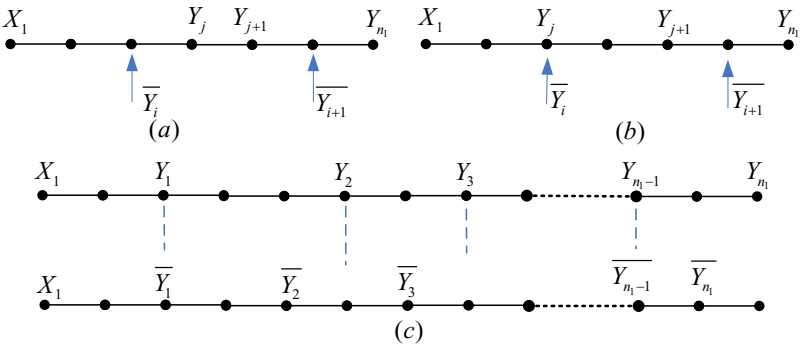


Fig. 4. Optimization of Least Overlap Algorithm

The inequation $\overline{Y_i} \leq Y_j < Y_{j+1} < \overline{Y_{i+1}}$ holds for at least a couple of $\overline{Y_i}, \overline{Y_{i+1}}$, as Fig. 4.(a) and Fig. 4.(b) show. If there is no such couple, we have $\overline{Y_{i+1}} \leq Y_{i+1}$ holds for $i \geq 1$. Hence, $|\overline{E_c}^*| \geq n_1$, as Fig. 4.(c) shows, which contradicts to the definition of $\overline{E_c}^*$. Therefore, $\overline{Y_i} \leq Y_j < Y_{j+1} < \overline{Y_{i+1}}$ holds for at least a couple of $\overline{Y_i}, \overline{Y_{i+1}}$.

In order to cover all the edges in E_{pr}, we have $\overline{X_{i+1}} \leq \overline{Y_i} + 1$. Because $\overline{Y_i} \leq Y_j < Y_{j+1} < \overline{Y_{i+1}}$ holds, we have $\overline{X_{i+1}} \leq \overline{Y_i} + 1 \leq Y_j + 1$. According to LO algorithm, $Y_{j+1} = \max_{x_\delta \leq Y_j + 1} \{y_\delta\}$. Because $\overline{X_{i+1}} \leq Y_j + 1$, we have $Y_{j+1} \geq \overline{Y_{i+1}}$, which contradicts to $Y_{j+1} < \overline{Y_{i+1}}$. Therefore, the assumption $|E_c{}^*| = n_2 < n_1$ doesn't hold. Hence, $\overline{E_{LO}}^*$ is the minimum one to cover all the edges in E_{pr} and also the one to cover all the edges in E_{ed}. □

From above all, if we have all the MHPS of same length, we can get the $\mathtt{C}_D^*$. If we have all the $\mathtt{C}_D^*$ with regards to different length D, $\mathtt{C}^* = \bigcup_D \mathtt{C}_D^*$.

4 Optimal Rescue Path Set

According to the analysis in the above sections, it needs four steps to get the optimal rescue paths:

(1) compute the shortest path;

(2) compute all the MHPs;

(3) delete all the replacement paths which are of the same length of the corresponding detour paths;

(4) compute the minimum joint replacement path set $\mathtt{C}_D^*$.

Based on the LO algorithm, we propose the Subtrees Connected Algorithm to compute the optimal rescue paths following the above four steps.

Subtrees Connected (SC) Algorithm:

Step 1: Compute the shortest path tree T_s(rooted at s) and T_t(rooted at t) by the Dijkstra algorithm [15]. As Fig. 5 shows, derive the distances from v_i to s and t, called $u(i)$ and $v(i)$, respectively. Cut T_s into p subtrees $\{T_s(i)|i = 1, \cdots, p\}$ along the edges in E_{sp}, where $T_s(i)$ is the subtree connected to v_i(see Fig.5 (a)). Let N_i denote the set of nodes on $T_s(i)$. Modify the tree T_s to be *Breadth-first type*(BF type), i.e. $\forall v_a \in N_i$ and $v_b \in N_j (i < j)$, $u(b) > u(a) + w(a, b)$. Update the $T_s(i)$ and N_i.

Step 2: Initialization. $\mathtt{C}^* = \{SP_G(s, t)\}$. Let $De(v_a) = i$ denote the depth of node $v_a \in N_i$. Let $A_i (i = 1, \cdots, p-1)$ denote the set of edges linking the nodes in N_i and $\overset{p}{\underset{j=i+1}{\cup}} N_j$, where $v_a \in N_i$, $v_b \in \overset{p}{\underset{j=i+1}{\cup}} N_j$ and the edges in E_{sp} are excluded from A_i. Let $l(v_i) = [l_1(i), l_2(i), l_3(i)] = [\infty, t, v_i]$ denote the temporary label of $v_i \in E$ and $Mark(i) = [Dist(i), x(i), y(i)] = [\infty, t, s]$ denote the permanent label of $v_i \in E_{sp}$. Let $h = 1$.

Step 3: Compute all the replacement paths with regards to every edge in E_{sp}. Suppose there are g_h edges in A_h, denote A_h as $\{e_q^{'} | q = 1, \cdots, g_h\}$. Let $q = 1$, $Detour = \infty$

Step 4: For $e_q^{'} = e(a, b)$, if $l_1(b) > u(a)+w(a, b)+v(b)$, $l(v_b) = [u(a)+w(a, b)+ v(b), v_a, v_b]$. If $u(a) + w(a, b) + v(b) < Detour$, $Detour = u(a) + w(a, b) + v(b)$.

- If $l_1(b) < Dist(h)$ holds or $l_1(b) = Dist(h)$ and $De(y(h)) < De(v_b)$ hold, $Mark(h) = l(v_b)$;
- If $De(v_b) > h + 1$, if $l_1(b) < Dist(h + 1)$ holds or $l_1(b) = Dist(h + 1)$ and $De(y(h + 1)) < De(v_b)$ hold, $Mark(h + 1) = l(v_b)$.

$q = q + 1$,

- if $q \leq g_h$, go to Step 4;
- if $q > g_h$, $h = h + 1$. If $Dist(h) = Detour$(the real-time detour path is also the replacement path), then $Mark(h) = [\infty, v_a, v_b]$(it will be deleted in Step 5). If $h < p$, go to Step 3; if $h = p$, go to Step 5.

Step 5: Compute the minimum joint replacement path sets. Sort the permanent labels $Mark(h)$ in increasing order of $Dist(h)$. If $Dist(h) = \infty$, delete $Mark(h)$. Partition the $Mark(h)$ with different $Dist(h)$ values into different sets $M_a (a = 1, \cdots, A)$.

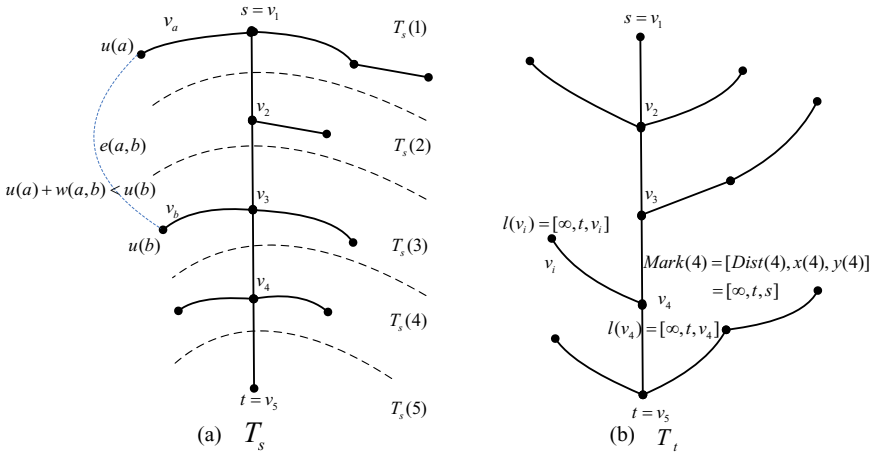


Fig. 5. Shortest path trees

For each $M_a(1 \le a \le A)$:

Step 5.1: Define the edge set $E^a_{ed} = \{e(h, h+1) \,|Mark(h) \in M_a\}$. $\forall Mark(h)$, $Mark(h^{'}) \in M_a$, if $Mark(h) = Mark(h')$, let $M_a = M_a - \{Mark(h^{'})\}$. For each $Mark(h) \in M_a$, when $x(h) = v_i$ and $y(h) = v_j$, denote $P_h = SP_G(s, x(h)) + e(x(h), y(h)) + SP_G(y(h), t)$ as $P_{i,j}$, and construct the corresponding TRES $E^a_{x_{ij},y_{ij}}$. Let $\mathtt{C}_D$ be $\{P_h|Mark(h) \in M_a\}$ and E^a_{TR} be $\{E^a_{x_{ij},y_{ij}}|Mark(h) \in M_a\}$.

Step 5.2: For E^a_{TR}, compute the minimum sub set E^*_a by LO algorithm. If $E^a_{x_{ij},y_{ij}} \in E^*_a$, $\mathtt{C}^* = \mathtt{C}^* \cup \{P_h\}$, where $P_h = P_{i,j}$.

Step 6: Output $\mathtt{C}^*$. Stop.

Lemma 3. *The minimum joint replacement path set $\mathtt{C}^*_D$ derived from the $\mathtt{C}_D$ in Step 5.1 is the same as that derived from all the MHPs corresponding to E^a_{ed}.*

Proof. In Step 1, T_s is modified to be Breadth-first type, i.e. $\forall v_a \in N_i$ and $v_b \in N_j (i < j)$, $u(b) > u(a) + w(a, b)$. For any $Mark(h)$, no $Mark^{'}(h) = [Dist^{'}(h), x^{'}(h), y^{'}(h)]$ satisfies the following conditions: $Dist^{'}(h) = Dist(h)$, $De(x^{'}(h)) < De(x(h))$ and $De(y(h)) = De(y^{'}(h))$, and the corresponding path $P^{'}_h$ passing node v_a(as Fig. 6 shows).

In Step 4, $Mark(h)$ will be updated only when $Dist(h)$ can be decreased or when $Dist(h)$ keeps the same while $De(y(h))$ increases. Therefore, for any $Mark(h)$, no $Mark^{'}(h) = [Dist^{'}(h), x^{'}(h), y^{'}(h)]$ satisfies the following conditions: $Dist^{'}(h) \le Dist(h)$, $De(x^{'}(h)) \le De(x(h))$ and $De(y(h)) \le De(y^{'}(h))$, as $P^{'}_{1h}$ and $P^{'}_{2h}$ in Fig. 6. Hence, the path P_h is an MHP.

However, there may exist $Mark^{'}(h) = [Dist^{'}(h), x^{'}(h), y^{'}(h)]$ satisfying the following conditions: $Dist^{'}(h) = Dist(h)$, $De(x^{'}(h)) < De(x(h))$ and $De(y^{'}(h)) < De(y(h))$. Let $i = De(x^{'}(h))$ $j = De(x^{'}(h))$, if $E^- = E_{ed} \cap \bar{E}_{i,j} = \phi$, P_h avoids no less edges in E_{ed} than $P^{'}_h$ does, and deleting $P^{'}_h$ will not effect the $\mathtt{C}^*_D$, see Fig .7(a); if $E^- \neq \phi$, $P^{'}_h$ must be the replacement path of this edge and involved

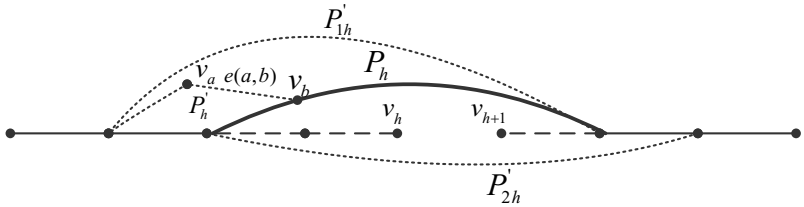


Fig. 6. Maximal replaced edge set

in the corresponding $\mathtt{C}_D$, see Fig .7(b). Hence, the missed MHP P'_h as Fig .7(a) shows will not have any effect the minimum joint replacement path set.

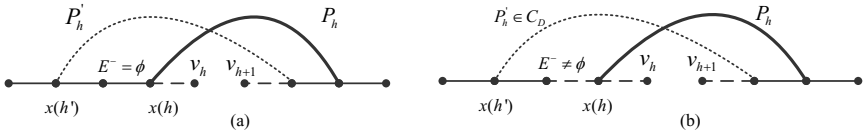


Fig. 7. The case when $De(y(h))$ decrease

Moreover, one MHP may be the replacement path with regards to several edges, i.e. $\forall Mark(i), Mark(j) \in M_a$, $Mark(i) = Mark(j)$. Hence, this operation does not miss any MHP.

From above all, the SC algorithm derive all the MHPs except these as Fig .7(a) shows. Because these MHPs has no effect on the $\mathtt{C}^*_D$, it holds that the $\mathtt{C}^*_D$ derived from the $\mathtt{C}_D$ computed in Step 5.1 is the same as that derived from all the MHPs corresponding to E^a_{ed}. The lemma follows.

□

Theorem 1. *Subtrees Connected Algorithm correctly computes the optimal rescue path set.*

Proof. We prove the correction of the algorithm from the two properties of optimal rescue path set, i.e. shortest time and minimum paths.

Shortest time: to ensure the track of the first arrival is the shortest path on the current network no matter which edge is blocked. Suppose the blocked edge is e_b,

Case 1: No blockage or $e_b \notin E_{sp}$. In this case, the vehicle that takes the shortest path is the first arrival.

Case 2: $e_b \in E_{sp}$ and $SP_{G-\{e_b\}}(s,t)$ overlaps $P(e_b)$. The vehicle that takes the shortest path will detour from the prefix node of e_b, and so its track is also the shortest path on the current network.

Case 3: $e_b \in E_{sp}$ and $SP_{G-\{e_b\}}(s,t)$ is different from $P(e_b)$. $Mark(b)$ is sorted in Step 5 and e_b is included in some E^a_{ed}. In Step 5.1, there is at least one TRES including e_b. The LO algorithm also ensures that there is at least one set $E^a_{x_i,y_i}$ in $\overline{E}^*_a$ covers e_b. It means that there is at least one replacement path with regards to e_b in $P_{N_{opt}-com}$.

Over all, there is at least one vehicle whose real track is the shortest path on the current network. Hence, the shortest time can be ensured.

Minimum paths: In Step 5, the replacement paths of the same length of the corresponding realtime detour paths are excluded from the path set. According to Lemmas 2 and 3, we can get all the minimum joint replacement path sets $E_a^*(1 \leq a \leq A)$ by the LO algorithm. It means that the redundant replacement paths are also excluded. Hence, there is no redundancy.

From above all, the SC algorithm can get the optimal rescue paths sets $\mathtt{C}^*$. □

Theorem 2. *The time complexity of Subtrees Connected Algorithm is* $O(m + n \log n)$.

Proof. Step 1 uses the Dijkstra algorithm implemented with Fibonacci heaps [14] to get the shortest path trees in $O(m + n \log n)$ time. It needs $O(n)$ time to compute the distances from s and t, $O(n)$ time to partition all the nodes into different sets, $O(m)$ time to modify T_s to the Breadth-first type and to update the N_i and $T_s(i)$.

Step 2 needs $O(m)$ time to initialize.

In Step 3, to get the replacement paths needs to update the $Mark(h)$ at most $2g_h$ times and to update the $l(v_i)$ at most g_h times. In Step 3 and Step 4, there are totally $\sum_{h=1}^{p-1} 3g_h < 3m$ update operation. Hence, it needs $O(m)$ time.

In Step 5, to sort the $Mark(i)$ needs $O(n \log n)$ time. For M_a, let $n_a = |M_a|$. In Step 5.1, to define E_{ed}^a needs at most n_a time, to exclude the duplicate elements in M_a needs at most $2n_a \log n_a$ time, to construct all the MHPs needs at most n_a time, and to construct the TRESs needs at most $4n_a \log n_a$ time. In Step 5.2, LO algorithm needs at most $n_a \log n_a + n_a$ time to compute the minimum sub set family E_a^* and at most n_a time to output the corresponding MHPs.

For all the $M_a(1 \leq a \leq A)$, there are totally $O(\sum_{a=1}^{A} (4n_a + 7n_a \log n_a)) = O(n \log n)$ times needed in Step 5.1-5.2.

Over all, the time complexity of SC algorithm is $O(m + n \log n)$. □

5 Conclusion

In this paper, we try to minimize the number of rescue vehicles to guarantee the fastest response to emergency even when some edge blockage might occur. We first proposed the definition of *joint replacement path*and *optimal rescue path set*. *SC* algorithm is presented to compute the optimal rescue path set, and its time complexity is proved to be $O(m + n \log n)$.

The optimal rescue path set is a path set for multiple rescue vehicles, which can realize the fastest response by dispatching the fewest rescue vehicles. Therefore, it is helpful for improving the distribution of the limited rescue resources as well as to minimize the emergency response time.

In this paper, we only consider the scenario that there is at most one edge is blocked. The scenario with multiple blockages is more practical. This scenario is

based on the definition of k most vital arcs, but even more complex than the k most vital arcs problem. It is because that it needs to compute all the $l(l \leq k)$ most vital arcs. However, the complexity of the k most vital arcs is still an open problem (see Malik [7]). Therefore, it is worthy to prove the complexity of the new scenario or study a more practical case in the future research. Furthermore, our study is based on the assumption that there is only one shortest path on graph G. The more general case with more than one shortest path in G is also an interesting research direction.

Acknowledgments. The authors would like to acknowledge the financial support of Grants (No.71071123, 61221063.) from NSF of China and (No.IRT1173) from PCSIRT of China.

References

1. Yen, J.Y.: Finding the K shortest loopless paths in a network. Management Science 17, 712–716 (1971)
2. Eppstein, D.: Finding the k shortest paths. SIAM Journal on Computing 28(2), 652–673 (1998)
3. Katoh, N., Ibaraki, T., Mine, H.: An efficient algorithm for K shortest simple paths. Networks 12(4), 411–427 (1982)
4. Akgun, V., Erkut, E., Batta, R.: On finding dissimilar paths. European Journal Of Operational Research 121(2), 232–246 (2000)
5. Adhari, H., Dreibholz, T., Becke, M.: Evaluation of concurrent multipath transfer over dissimilar paths. In: Proceedings of the 25th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA), pp. 708–714 (2011)
6. Dell'Olmo, P., Gentili, M., Scozzari, A.: On finding dissimilar pareto-optimal paths. European Journal Of Operational Research 162(1), 70–82 (2007)
7. Malik, K., Mittal, A.K., Gupta, S.K.: The k most vital arcs in the shortest path problem. Operations Research Letters 8(4), 223–227 (1989)
8. Nardelli, E., Proiett, G., Widmayer, P.: A faster computation of the most vital edge of a shortest path. Information Processing Letter 79(2), 81–85 (2001)
9. Emek, Y., Peleg, D., Rodity, L.: A near-linear-time algorithm for computing replacement paths in planar directed graphs. ACM Transactions on Algorithms 6(4), Article 64 (2010)
10. Wulff-Nilsen, C.: Solving the replacement paths problem for planar directed graphs in $O(n \log n)$ time. In: Proceedings of the 21th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 756–765 (2010)
11. Nardelli, E., Proietti, G., Widmayer, P.: Finding the detour-critical edge of a shortest path between two nodes. Information Processing Letters 67(1), 51–54 (1998)
12. Xiao, P., Xu, Y., Su, B.: Finding an anti-risk path between two nodes in undirected graphs. Journal of Combinatorial Optimization 17(3), 235–246 (2009)
13. Hershberger, J., Suri, S.: Vickrey prices and shortest paths: What is an edge worth? In: Proceedings of the 42nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2001), pp. 252–259 (2001)
14. Fredman, M., Tarjan, R.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 34(3), 596–615 (1987)
15. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)

Expected Computations on Color Spanning Sets*

Chenglin Fan[1], Jun Luo[1], Farong Zhong[2], and Binhai Zhu[3]

[1] Shenzhen Institutes of Advanced Technology
Chinese Academy of Sciences, Shenzhen, China
[2] Zhejiang Normal University, China
[3] Department of Computer Science, Montana State University
Bozeman, MT 59717, USA
{cl.fan,jun.luo}@siat.ac.cn,
zrf@zjnu.cn, bhz@cs.montana.edu

Abstract. Given a set of n points, each is painted by one of the k given colors, we want to choose k points with distinct colors to form a color spanning set. For each color spanning set, we can construct the convex hull and the smallest axis-aligned enclosing rectangle, etc. Assume that each point is chosen independently and identically from the subset of points of the same color, we propose an $O(n^2 \log n)$ time algorithm to compute the expected area of convex hulls of the color spanning sets and an $O(n^2 \log n)$ time algorithm to compute the expected perimeter of convex hulls of the color spanning sets. For the expected perimeter (resp. area) of the smallest perimeter (resp. area) axis-aligned enclosing rectangles of the color spanning sets, we present an $O(n \log n)$ (resp. $O(n^2)$) time algorithm. We also propose an approximation algorithm to compute the expected diameter of the color spanning sets.

Keywords: Expected Value, Imprecise Data, Computational. Geometry.

1 Introduction

Most of the classic algorithms in computational geometry are based on the assumption that the locations of input points are known exactly. But, in practice, that is not always the case. We can, more often than not, only obtain the data varying within some ranges. For example, the locations of a moving object have some uncertain properties [3]. Moreover, location data obtained from physical devices are inherently imprecise due to the measurement error, sampling error and network latency [8, 9]. Location privacy protection is another issue which may lead to imprecise data [2, 4, 7].

Each imprecise point p can be modelled by a continuous region ϕ or even a set ψ of discrete points. That means p could be anywhere in ϕ or any one

* This research has been partially funded by the International Science & Technology Cooperation Program of China (2010DFA92720) and NSF of China under poject 11271351.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 130–141, 2013.

point of ψ but its exact location is not known. For a set of imprecise points $P = \{p_1, p_2, ..., p_n\}$, they can be modelled by a set $\Phi = \{\phi_1, \phi_2, ..., \phi_n\}$ of continuous regions or a set $\Psi = \{\psi_1, \psi_2, ..., \psi_n\}$ of sets of discrete points.

In the database community, a similar framework has been researched under a different name – which is called "uncertain data" and, within the framework, each color is called an "object" (the different instances of a color correspond to the different possible instances of the uncertain object). In [11–13, 22, 24, 25], there are obtained results that pertain to geometric problems in this framework.

The problem above is often referred to as the color-spanning problem, where each imprecise point is modelled by a set of discrete points with each set of discrete points painted by one distinct color. In general, the color spanning problem is to select exactly one point from each color such that certain properties (e.g. area, distance, perimeter and so on) of some geometric structures (e.g. convex hulls, minimum spanning trees and so on) based on the selected points with different colors are minimized or maximized. In the following descriptions, we assume that there are n points with k colors for the sake of notation consistency. Zhang *et al.* [10] proposed a brute force algorithm to address the minimum diameter color-spanning set problem (MDCS). The running time is $O(n^k)$. Fleischer and Xu [6] showed that the MDCS problem can be solved in polynomial time for the L_1 and L_∞ metrics, while it is NP-hard for all other L_p metrics, even in two dimensions. They also gave an efficient algorithm to compute a constant factor approximation.

Abellanas *et al.* [1] showed that the Farthest Color Voronoi Diagram (FCVD) is of complexity $\Theta(nk)$ if $k \leq n/2$. Then they proposed algorithms to construct FCVD, the smallest color-spanning circle based on FCVD, the smallest color-spanning rectangle and the narrowest color-spanning strip of arbitrary orientation. In [5], Das *et al.* proposed an algorithm for identifying the smallest color-spanning corridor in $O(n^2 \log n)$ time and an algorithm for identifying the smallest color-spanning rectangle of arbitrary orientation with an $O(n^3 \log k)$ running time. In [20]. Ju *et al.* studied several other problems regarding color-spanning sets, like diameter and minimum spanning tree, etc.

In this paper, instead of computing the maximum or minimum value of some geometric properties, we study expected values of some geometric properties, which has received much less attention in the algorithm community. However, sometimes expected values are more meaningful than extreme values — as the former can be used to estimate the future events. For example, in database community, people studied a lot of problems on probabilistic databases [14] such as histogram building [15], clustering [16] and indexing [17].

Regarding the geometric properties covered in this paper, it is NP-hard to compute the largest or smallest (area or perimeter) convex hull of imprecise input at general positions [18,20]. Hence computing the expected area or perimeter of convex hull of imprecise input is a good choice in reality. The discrete version of imprecise input are also called indecisive point sets in [19]. Jørgensen *et al.* [19] studied computing distributions of geometric functions such as the radius of the smallest enclosing ball and the diameter, and showed that the distribution of

the radius of the smallest enclosing ball can be calculated exactly in polynomial time, but computing the same distribution for the diameter is #P-hard. They generalized the polynomial time algorithm to all LP-type problems, which does not work for any property of the convex hull (e.g. area or perimeter) because it does not have a constant combinatorial dimension. However, we show that the expected perimeter or area of convex hull of indecisive point sets can be computed in polynomial time in this paper. Kamousi *et al.* [23] studied the problem of Stochastic Minimum Spanning Trees in Euclidean Spaces, and showed the stochastic MST problem is #P-hard for any dimension larger or equal than 2 and proposed approximation algorithms. Paper [23] also mentions that the expected lengths of some geometry structure (convex hull,Delaunay triangulation,Gabriel graph or relative neighborhood graph *etc*) are not hard to compute, but no details are given in that paper. Jørgensen *et al.* [19] also showed that the LP-type framework can compute distribution of the smallest axis-aligned bounding box, by perimeter, of indecisive point sets in $O((nk)^5)$ time, where n is the total number of points and k is the total number of colors. The expected value of the smallest axis-aligned bounding box can be computed after the distribution is calculated in their LP-type framework, but the $O((nk)^5)$ running time is too much. In this paper, we show that the expected value of perimeter (resp. area) for the smallest axis-aligned bounding box can be computed in $O(n \log n)$ (resp. $O(n^2)$) time, significantly improving the previous bound. Since the problem of expected diameter of indecisive points is difficult to solve [19], we give a $2/\sqrt{3}$ approximation algorithm running in $O((nk)^4)$ time.

We formally define the five problems to be studied in section 2. The five algorithms are given in respective sections. We give conclusions in the last section.

2 Problem Definition

Given a set of n points, each point is painted with one of the k given colors, we want to choose k points with distinct colors to form a color spanning set. For each color spanning set we can construct convex hull and axis-aligned (enclosing) rectangle. Suppose that each point is chosen independently and identically from the points of the same color. Assume that there are m_i points painted with the i-th color such that $n = \sum_{i=1}^{k} m_i$, then there are $M = \prod_{i=1}^{k} m_i$ possible ways to choose k color spanning points. Note that M could be exponential in the worst case.

In this paper, we assume that the probability of appearance for each point in color c_i is equal, namely $1/m_i$, and we want to compute five expected values as follows:

Problem 1. The expected value of perimeter for M convex hulls.

Problem 2. The expected value of area for M convex hulls.

Problem 3. The expected value of perimeter for M smallest perimeter axis-aligned (enclosing) rectangles.

Problem 4. The expected value of area for M smallest area axis-aligned (enclosing) rectangles.

Problem 5. The expected value of diameter for M color spanning sets.

For a convex hull C, we use $Per(C)$ and $Area(C)$ to denote the perimeter and area of C respectively. For an edge e, we use $L(e)$ to represent its length. The set of M convex hulls is denoted as $S_C = \{C_1, C_2, ..., C_M\}$. The probability for constructing each C_i is $P(C_i) = \frac{1}{M}$.

3 Algorithm for Problem 1

The expected value of perimeter for M convex hulls can be defined as follows:

$$E_{cp} = \sum_{i=1}^{M} (P(C_i) * Per(C_i)).$$

If we use $e \in C_i$ to represent an edge of C_i, then

$$E_{cp} = \frac{1}{M} \sum_{\forall C_i \in S_C} \sum_{\forall e \in C_i} L(e)$$

$$= \frac{1}{M} \sum_{\forall e \in C_i} \sum_{\forall C_i \in S_C} L(e).$$

Let $P(e) = \frac{1}{M} \sum_{\forall C_i \in S_C} exist(e, C_i)$ be the probability of e being an edge of convex hull, where the function $exist(e, C_i)$ returns 1 if e is one of the edges of C_i and it returns 0 otherwise. Then, we obtain

$$E_{cp} = \sum_{\forall e} P(e) * L(e).$$

Let $l(e)$ be the extended line of e. According to the property of convex hull, e is an edge of convex hull iff all the points appear on one side of $l(e)$ or on e. Let $m_i^{upper(e)}$ be the number of points of the i-th color on the upper side of $l(e)$ or on e. Let $m_i^{bottom(e)}$ be the number of points of the i-th color on the bottom side of $l(e)$ or on e. Suppose that the two endpoints of e are painted with the j_1-th and j_2-th colors respectively. Then

$$P(e) = \frac{1}{M} \left(\frac{\prod_{i=1}^{k} m_i^{upper(e)}}{m_{j_1}^{upper(e)} \times m_{j_2}^{upper(e)}} + \frac{\prod_{i=1}^{k} m_i^{bottom(e)}}{m_{j_1}^{bottom(e)} \times m_{j_2}^{bottom(e)}} \right) \quad (1)$$

where the first (second) part in the parenthesis represents the number of possible convex hulls with e as the bottommost (uppermost) edge.

We now present an algorithm running in $O(n^2 \log n)$ time to compute E_{cp}. For a point a, we can sort all other $n-1$ points by the angles around a (see Fig. 1).

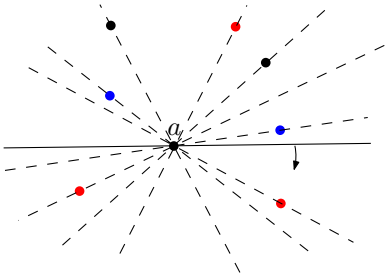


Fig. 1. Illustration of computing E_{cp}

Then we draw a line horizontally through a and rotate it around a clockwise until it hits a point b of a color different from a. Let $e = ab$. We can count the number of points for different colors in the upper and bottom side of $l(e)$. Consequently, we can obtain $m_i^{upper(e)}$ and $m_i^{bottom(e)}$ to compute $P(e)$, where $i = 1, ..., k$. Note that $m_{j_1}^{upper(e)}, m_{j_2}^{upper(e)}, m_{j_1}^{bottom(e)}, m_{j_2}^{bottom(e)}$ are in $m_i^{upper(e)}$ and $m_i^{bottom(e)}$. We continue to rotate $l(e)$ clockwise until it hits another point c with the j_3-th color. Let $e = ac$. If $j_3 \neq j_1$, then $m_{j_3}^{upper(e)}$ ($m_{j_3}^{bottom(e)}$) are increased or decreased by one depending on whether c enters or leaves upper (bottom) side of $l(e)$ respectively. All other m_i's ($i \neq j_3$) do not change. Then we can compute $P(e)$. If $j_3 = j_1$, $m_{j_3}^{upper(e)}$ and $m_{j_3}^{bottom(e)}$ are updated as before but we do not have to compute $P(e)$. The process is stopped when the sweep line rotates 180°.

For the running time, the sorting step takes $O(n \log n)$ time for the fixed point a. For $e = ab$, $m_i^{upper(e)}$, $m_i^{bottom(e)}$ and $P(e)$, where $i = 1, ..., k$, can be computed in $O(n)$ time. After that, each update only takes constant time, therefore the whole sweeping process takes $O(n)$ time. Since we have $O(n)$ possible a's, the total running time is $O(n^2 \log n)$.

Theorem 1. *The expected value of perimeter for M convex hulls can be computed in $O(n^2 \log n)$ time.*

4 Algorithm for Problem 2

The expected value of area for M convex hulls can be defined as follows:

$$E_{ca} = \sum_{i=1}^{M} (P(C_i) * Area(C_i)).$$

We draw a horizontal line H below all the n points. For an edge $e \in C_i$, we define $Area(e)$ as the area of trapezoid formed by e, H and two vertical line segments

projected from the two endpoints of e to H. Then

$$Area(C_i) = \sum_{\forall e \in upper\ hull\ of\ C_i} Area(e) - \sum_{\forall e \in lower\ hull\ of\ C_i} Area(e)$$

We know that $e \in$ *upper hull of* C_i iff all points of C_i are below $l(e)$ and $e \in$ *lower hull of* C_i iff all points are above $l(e)$. Therefore

$$E_{ca} = \sum_{\forall e} P(e) * Area(e),$$

where

$$P(e) = \frac{1}{M}\left(\frac{\prod_{i=1}^{k} m_i^{bottom(e)}}{m_{j_1}^{bottom(e)} \times m_{j_2}^{bottom(e)}} - \frac{\prod_{i=1}^{k} m_i^{upper(e)}}{m_{j_1}^{upper(e)} \times m_{j_2}^{upper(e)}}\right) \tag{2}$$

This is different from equation 1, but the meanings of notations and the computing process are the same as in problem 1.

Theorem 2. *The expected value of area for M convex hulls can be computed in $O(n^2 \log n)$ time.*

5 Algorithm for Problem 3

The expected value of perimeter for M smallest perimeter axis-aligned rectangles can be defined as follows:

$$E_{rp} = \frac{1}{M} \sum_{\forall R_i \in S_R} Per(R_i),$$

where $S_R = \{R_1, R_2, ..., R_M\}$ is the set of M smallest perimeter axis-aligned rectangles enclosing k points of different colors.

For the n input points, we can sort them according to the y-coordinates to obtain the sorted sequence $p_1^v, p_2^v, ..., p_n^v$ from top to bottom. Similarly we can obtain the sorted sequence $p_1^h, p_2^h, ..., p_n^h$ from left to right. We can draw vertical and horizontal lines through all points. For point p, let $x.p$ and $y.p$ represent x and y coordinates of p respectively.

Lemma 1. *Each R_i either covers the whole horizontal interval $[x.p_i^h, x.p_{i+1}^h]$ $(1 \le i < n)$ or does not cover any part of it (except possibly $x.p_i^h$ or $x.p_{i+1}^h$). Similarly R_i either covers the whole vertical interval $[y.p_i^v, y.p_{i+1}^v]$ $(1 \le i < n)$ or does not cover any part of it (except possibly $y.p_i^v$ or $y.p_{i+1}^v$).*

Proof. We only need to prove the former case since the proof of the latter case is similar. We prove it by contradiction. Suppose some R_i only covers a part of $(x.p_j^h, x.p_{j+1}^h)$, then we can move the left or right edge to $x.p_j^h$ or $x.p_{j+1}^h$) to obtain an axis-aligned rectangle enclosing original k points of a smaller perimeter. This contradicts the fact of R_i being the smallest perimeter axis-aligned rectangle.

□

Suppose that R_i covers the horizontal interval $[x.p_a^h, x.p_b^h]$ and the vertical interval $[y.p_c^v, y.p_d^v]$. Then

$$E_{rp} = \frac{2}{M} \sum_{\forall R_i \in S_R} ((x.p_b^h - x.p_a^h) + (y.p_d^v - y.p_c^v)).$$

From Lemma 1, we know that for a horizontal interval $[x.p_i^h, x.p_{i+1}^h]$, it is covered by R_i iff for k points of different colors enclosed by R_i, some of the k points are to the left of $x.p_i^h$ and some of k points are to the right of $x.p_{i+1}^h$. In other words, $[x.p_i^h, x.p_{i+1}^h]$ is not covered by R_i iff all the k points are to the left of $x.p_i^h$ or all k points are to the right of $x.p_{i+1}^h$. Similar properties hold for $[y.p_i^v, y.p_{i+1}^v]$. Let $P(h_i)$ be the probability that the horizontal interval $[x.p_i^h, x.p_{i+1}^h]$ is covered by some rectangle in S_R and $P(v_i)$ be the probability that the vertical interval $[y.p_i^v, y.p_{i+1}^v]$ is covered by some rectangle in S_R. Then

$$E_{rp} = \frac{2}{M} \sum_{i=1}^{n-1} (P(h_i) \times (x.p_{i+1}^h - x.p_i^h) + P(v_i) \times (y.p_{i+1}^v - y.p_i^v))$$

$$= \frac{2}{M} \sum_{i=1}^{n-1} ((1 - \overline{P}(h_i)) \times (x.p_{i+1}^h - x.p_i^h) + (1 - \overline{P}(v_i)) \times (y.p_{i+1}^v - y.p_i^v)),$$

where $\overline{P}(h_i)$ represents the probability that the horizontal interval $[x.p_i^h, x.p_{i+1}^h]$ is not covered by some rectangle in S_R and $\overline{P}(v_i)$ represents the probability that the vertical interval $[y.p_i^v, y.p_{i+1}^v]$ is not covered by some rectangle in S_R. Let $m_{i,j}^{left}$ be the number of points with the j-th color to the left of $x.p_i^h$ and $m_{i,j}^{right}$ be the number of points with the j-th color to the right of $x.p_i^h$. Let $m_{i,j}^{upper}$ be the number of points with the j-th color above $y.p_i^v$ and $m_{i,j}^{lower}$ be the number of points with the j-th color below $y.p_i^v$. Then

$$\overline{P}(h_i) = \frac{1}{M} (\prod_{j=1}^{k} m_{i,j}^{left} + \prod_{j=1}^{k} m_{i+1,j}^{right})$$

$$\overline{P}(v_i) = \frac{1}{M} (\prod_{j=1}^{k} m_{i,j}^{upper} + \prod_{j=1}^{k} m_{i+1,j}^{lower})$$

To compute $\overline{P}(h_i)$, we can scan the horizontal intervals from left to right. Initially, $m_{1,j}^{left}$ and $m_{2,j}^{right}$ can be computed in $O(n)$ time. Then each $m_{i,j}^{left}$ and $m_{i+1,j}^{right}$ can be updated in constant time. Thus all $\overline{P}(h_i)$ $(i = 1, ..., n)$ and E_{rp} can be computed in linear time. The whole algorithm still takes $O(n \log n)$ time since we need to sort all the n points vertically and horizontally at the beginning.

Theorem 3. *The expected value of perimeter for M smallest perimeter axis-aligned rectangles can be computed in $O(n \log n)$ time.*

6 Algorithm for Problem 4

The expected value of area for M smallest area axis-aligned rectangles can be defined as follows:

$$E_{ra} = \frac{1}{M} \sum_{\forall R_i \in S_R} Area(R_i),$$

where $S_R = R_1, R_2, ..., R_M$ is the set of M smallest area axis-aligned rectangles enclosing k points of different colors.

Again, we can draw n vertical lines and n horizontal lines through n points. Those $2n$ lines form $(n-1)^2$ (bounded) rectangles. Let r_{ij} represent the rectangle with horizontal interval $[x.p_i^h, x.p_{i+1}^h]$ and vertical interval $[y.p_j^v, y.p_{j+1}^v]$. Then

$$E_{ra} = \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} (1 - \overline{P}(r_{ij})) \times Area(r_{ij}),$$

where $\overline{P}(r_{ij})$ is the probability that r_{ij} is not covered by some rectangle in S_R.

Similarly to the proof of Lemma 1, we can obtain the following lemma:

Lemma 2. *Each R_i either covers the whole r_{ij} or does not cover any part of it (except possibly the corners).*

From the above lemma, we know that r_{ij} is not covered by R_i iff for k points of different colors enclosed by R_i, all k points are above $y.p_j^v$, or all k points are below $y.p_{j+1}^v$, or all k points are to the left of $x.p_i^h$, or all k points are to the right of $x.p_{i+1}^h$. For each rectangle r_{ij}, we draw four lines through four edges of r_{ij}. Those four lines partition the plane into nine regions marked as $r_{ij}^N, r_{ij}^W, r_{ij}^E, r_{ij}^S, r_{ij}^{NW}, r_{ij}^{NE}, r_{ij}^{SW}, r_{ij}^{SE}$ and r_{ij} respectively (see Fig. 2). Let P_{ij}^{NE} be the probability that all k points are in r_{ij}^{NE}, P_{ij}^{SE} be the probability that all k points are in r_{ij}^{SE}, P_{ij}^{NW} be the probability that all k points are in r_{ij}^{NW}, P_{ij}^{SW} be the probability that all k points are in r_{ij}^{SW}, P_{ij}^{N} be the probability that all k points are in $r_{ij}^{NW} \bigcup r_{ij}^{N} \bigcup r_{ij}^{NE}$, P_{ij}^{S} be the probability that all k points are in $r_{ij}^{SW} \bigcup r_{ij}^{S} \bigcup r_{ij}^{SE}$, P_{ij}^{W} be the probability that all k points are in $r_{ij}^{SW} \bigcup r_{ij}^{W} \bigcup r_{ij}^{NW}$, P_{ij}^{E} be the probability that all k points are in $r_{ij}^{SE} \bigcup r_{ij}^{E} \bigcup r_{ij}^{NE}$. Then

$$\overline{P}(r_{ij}) = 1 - (P_{ij}^E + P_{ij}^W + P_{ij}^N + P_{ij}^S - P_{ij}^{NE} - P_{ij}^{SE} - P_{ij}^{NW} - P_{ij}^{SW})$$

Let $m_{ij,j'}^N$ be the number of points with the j'-th color in r_{ij}^N. Similarly, we can define $m_{ij,j'}^E, m_{ij,j'}^W, m_{ij,j'}^S, m_{ij,j'}^{NE}, m_{ij,j'}^{NW}, m_{ij,j'}^{SE}, m_{ij,j'}^{SW}$. Then

$$P_{ij}^E = \frac{1}{M} \prod_{j'=1}^{k} (m_{ij,j'}^E + m_{ij,j'}^{NE} + m_{ij,j'}^{SE}),$$

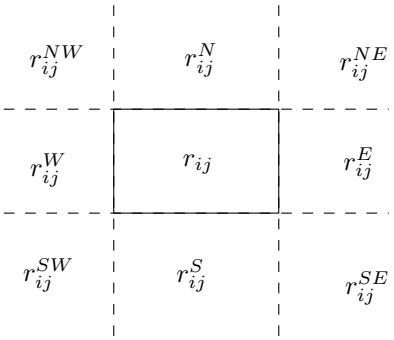


Fig. 2. Partition of the plane according four edges of r_{ij}

$$P_{ij}^{W} = \frac{1}{M} \prod_{j'=1}^{k} (m_{ij,j'}^{W} + m_{ij,j'}^{NW} + m_{ij,j'}^{SW}),$$

$$P_{ij}^{N} = \frac{1}{M} \prod_{j'=1}^{k} (m_{ij,j'}^{N} + m_{ij,j'}^{NE} + m_{ij,j'}^{NW}),$$

$$P_{ij}^{S} = \frac{1}{M} \prod_{j'=1}^{k} (m_{ij,j'}^{S} + m_{ij,j'}^{SE} + m_{ij,j'}^{SW}),$$

$$P_{ij}^{NE} = \frac{1}{M} \prod_{j'=1}^{k} (m_{ij,j'}^{NE}),$$

$$P_{ij}^{NW} = \frac{1}{M} \prod_{j'=1}^{k} (m_{ij,j'}^{NW}),$$

$$P_{ij}^{SE} = \frac{1}{M} \prod_{j'=1}^{k} (m_{ij,j'}^{SE}),$$

$$P_{ij}^{SW} = \frac{1}{M} \prod_{j'=1}^{k} (m_{ij,j'}^{SW}).$$

We can scan the rectangle r_{ij} from left to right and then from top to bottom. Initially, $m_{11,j'}^{N}, m_{11,j'}^{E}, m_{11,j'}^{W}, m_{11,j'}^{S}, m_{11,j'}^{NE}, m_{11,j'}^{NW}, m_{11,j'}^{SE}, m_{11,j'}^{SW}$ can be computed in linear time. Each update on $m_{ij,j'}^{N}$, $m_{ij,j'}^{E}$, $m_{ij,j'}^{W}$, $m_{ij,j'}^{S}$, $m_{ij,j'}^{NE}$, $m_{ij,j'}^{NW}$, $m_{ij,j'}^{SE}$, $m_{ij,j'}^{SW}$ can be done in constant time. All $P(r_{ij})$ $(i, j = 1, ..., n)$ can be computed in $O(n^2)$ time. Therefore E_{ra} can be solved in $O(n^2)$ time.

Theorem 4. *The expected value of perimeter for M smallest area axis-aligned rectangles can be computed in $O(n^2)$ time.*

7 Algorithm for Problem 5

The expected value of diameter for M color spanning sets can be defined as follows:

$$E_D = \sum_{i=1}^{M} (P(S_i) * D(S_i)),$$

where S_i is a color spanning set, $P(S_i) = \frac{1}{M}$ is the probability that S_i is a color spanning set, and $D(S_i)$ is the diameter of S_i. Then we have

$$E_D = \frac{1}{M} \sum_{i=1}^{M} D(S_i).$$

The exact expected value is not easy to compute as computing distribution for the diameter is #P-hard [19].

Theorem 5. *The expected value of diameter for M color spanning sets in the plane can be $2/\sqrt{3}$ approximated in polygonal time.*

Proof. We use the diameter of the smallest enclosing ball of color spanning sets to approximate the diameter of color spanning sets. The distribution of the radius of the smallest enclosing ball can be calculated exactly in $O((nk)^4)$ time [19], where n is the total number of points and k is the total number of sets (colors), and there are at most $O((nk)^3)$ discrete values of the distribution. The diameter of color spanning set can be $2/\sqrt{3}$ approximated by the diameter of smallest enclosing ball in the plane according to paper [6]. We sum the product of discrete value and its responding probability of distribution, which takes $O((nk)^3)$ time. And the theorem is proven. □

8 Conclusions

We propose an $O(n^2 \log n)$ time algorithm to compute the expected area (perimeter) of convex hulls of the color spanning sets. For the expected area (resp. perimeter) of the smallest area (resp. perimeter) axis-aligned enclosing rectangles of the color spanning sets, it can be computed in $O(n^2)$ (resp. $O(n \log n)$) time. We also propose a simple approximation algorithm to compute the expected diameter of the color spanning sets. For the future work, it will be interesting to see whether our technique can be applied to other expected value such as expected distance of the closest pair of M color spanning sets and so on. Those problems are also interesting in high dimensional space.

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. Beresford, A.R., Stajano, F.: Location privacy in pervasive computing. IEEE Pervasive Computing 2(1), 46–55 (2003)
3. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Querying imprecise data in moving object environments, knowledge and data engineering. IEEE Transactions on Knowledge and Data Engineering 16(9), 1112–1127 (2004)
4. Cheng, R., Zhang, Y., Bertino, E., Prabhakar, S.: Preserving user location privacy in mobile data management infrastructures. In: Danezis, G., Golle, P. (eds.) PET 2006. LNCS, vol. 4258, pp. 393–412. Springer, Heidelberg (2006)
5. Das, S., Goswani, P.P., Nandy, S.C.: Smallest color-spanning object revised. International Journal of Computational Geometry and Applications 19(5), 457–478 (2009)
6. Fleischer, R., Xu, X.: Computing Minimum Diameter Color-Spanning Sets. In: Lee, D.-T., Chen, D.Z., Ying, S. (eds.) FAW 2010. LNCS, vol. 6213, pp. 285–292. Springer, Heidelberg (2010)
7. Gedik, B., Liu, L.: A customizable k-anonymity model for protecting location privacy. In: Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS 2005), pp. 620–629 (2005)
8. Pfoser, D., Jensen, C.S.: Capturing the uncertainty of moving-object representations. In: Güting, R.H., Papadias, D., Lochovsky, F.H. (eds.) SSD 1999. LNCS, vol. 1651, pp. 111–131. Springer, Heidelberg (1999)
9. Sistla, A.P., Wolfson, O., Chamberlain, S., Dao, S.K.: Querying the uncertain position of moving objects. In: Etzion, O., Jajodia, S., Sripada, S. (eds.) Dagstuhl Seminar 1997. LNCS, vol. 1399, pp. 310–337. Springer, Heidelberg (1998)
10. 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)
11. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic Skylines on Uncertain Data. In: VLDB 2007, pp. 15–26 (2007)
12. Cheema, M.A., Lin, X., Wang, W., Zhang, W., Pei, J.: Probabilistic Reverse Nearest Neighbor Queries on Uncertain Data. IEEE Trans. Knowl. Data Eng. 22(4), 550–564 (2010)
13. Yuen, S.M., Tao, Y., Xiao, X., Pei, J., Zhang, D.: Superseding Nearest Neighbor Search on Uncertain Spatial Databases. IEEE Trans. Knowl. Data Eng. 22(7), 1041–1055 (2010)
14. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. VLDB J. 16(4), 523–544 (2007)
15. Cormode, G., Garofalakis, M.: Histograms and Wavelets on Probabilistic Data. IEEE Trans. on Knowl. and Data Eng. 22(8), 1142–1157 (2010)
16. Cormode, G., McGregor, A.: Approximation algorithms for clustering uncertain data. In: Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 191–200 (2008)
17. Tao, Y., Cheng, R., Xiao, X., Ngai, W.-K., Kao, B., Prabhakar, S.: Indexing multidimensional uncertain data with arbitrary probability density functions. In: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 922–933 (2005)
18. Löffler, M., van Kreveld, M.: Largest and smallest tours and convex hulls for imprecise points. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 375–387. Springer, Heidelberg (2006)

19. Jørgensen, A., Löffler, M., Phillips, J.M.: Geometric computations on indecisive points. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 536–547. Springer, Heidelberg (2011)
20. Ju, W., Fan, C., Luo, J., Zhu, B., Daescu, O.: On Some Geometric Problems of Color-Spanning Sets. Journal of Combinatorial Optimization (2012), doi:10.1007/s10878-012-9458-y
21. Goodman, J., O'Rourke, J.: Handbook of discrete and computational geometry. Chapman & Hall/CRC (2004)
22. Afshani, P., Agarwal, P.K., Arge, L., Larsen, K.G., Phillips, J.M.: (Approximate) uncertain skylines. In: Proceedings of the 14th International Conference on Database Theory, pp. 186–196 (2011)
23. Kamousi, P., Chan, T.M., Suri, S.: Stochastic minimum spanning trees in Euclidean spaces. In: Proceedings of the 27th Annual ACM Symposium on Computational Geometry, pp. 65–74 (2011)
24. Agarwal, P.K., Cheng, S.W., Yi, K.: Range searching on uncertain data. ACM Transactions on Algorithms 8(4), 43 (2012)
25. Agarwal, P.K., Efrat, A., Sankararaman, S., Zhang, W.: Nearest-neighbor searching under uncertainty. In: Proceedings of the 31st Symposium on Principles of Database Systems, pp. 225–236 (2012)

Independent Domination: Reductions from Circular- and Triad-Convex Bipartite Graphs to Convex Bipartite Graphs*

Min Lu[1], Tian Liu[1], and Ke Xu[2]

[1] Key Laboratory of High Confidence Software Technologies, Ministry of Education, School of Electronic Engineering and Computer Science, Peking University, Beijing 100871, China
lt@pku.edu.cn

[2] National Lab. of Software Development Environment, Beihang University, Beijing 100191, China
kexu@nlsde.buaa.edu.cn

Abstract. An *independent dominating set* in a graph is a subset of vertices, such that no edge has both ends in the subset, and each vertex either itself is in the subset or has a neighbor in the subset. In a *convex bipartite* (*circular convex bipartite*, *triad convex bipartite*, respectively) graph, there is a linear ordering (a circular ordering, a triad, respectively) defined on one class of vertices, such that for every vertex in the other class, the neighborhood of this vertex is an interval (a circular arc, a subtree, respectively), where a *triad* is three paths with a common end. The problem of finding a minimum independent dominating set, called *independent domination*, is known $\mathcal{NP}$-complete for bipartite graphs and tractable for convex bipartite graphs. In this paper, we make polynomial time reductions for independent domination from triad- and circular-convex bipartite graphs to convex bipartite graphs.

Keywords: Independent domination, circular convex bipartite graph, triad convex bipartite graph, polynomial time reduction.

1 Introduction

An *independent dominating set* in a graph is a subset of vertices, such that the subset is an independent set, and every vertex in the graph either itself is in the subset or has a neighbor in the subset. The problem of finding a minimum independent dominating set, called *independent domination*, is $\mathcal{NP}$-complete for *chordal bipartite* graphs, but polynomial time solvable for *convex bipartite* graphs [3]. In a *convex bipartite* graph [7,3,2], there is a linear ordering defined on one class of vertices, such that for every vertex in another class, the neighborhood of this vertex is an interval. In a *chordal bipartite* graph [6], every cycle of length at least *six* has a chord, where a *chord* of a cycle on a graph is an edge between two vertices of the cycle but the edge itself is not a part of the cycle.

* Partially supported by National 973 Program of China (Grant No. 2010CB328103).

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 142–152, 2013.

Beside convex bipartite graphs and chordal bipartite graphs, there are other interesting bipartite graph classes, such as *circular convex bipartite* graphs [11] and *triad convex bipartite* [10,9] graphs, etc, see Figure 1.

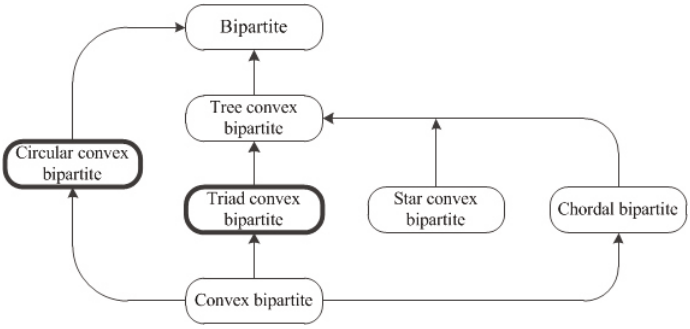


Fig. 1. Various bipartite graph classes and their inclusions

In a *circular convex bipartite* graph [11], there is a circular ordering defined on one class of vertices, such that for every vertex in another class, the neighborhood of this vertex is a circular arc. Circular convex bipartite graphs are natural models for scheduling problems. For example, the available working hours of a worker is usually a consecutive period of hours. A group of workers and their available hours can be modeled by a circular convex bipartite graph [11]. For a long time, complexity results for circular convex bipartite graphs are scarce. Maximum matching and Hamiltonian cycle and path are known linear time solvable for circular-convex bipartite graphs [11]. The complexity of independent domination for circular-convex bipartite graphs is unknown before. In this paper, we show that independent domination is *polynomial time* solvable for *circular convex bipartite* graphs.

In a *tree convex bipartite* graph [8,9], there is a tree defined on one class of vertices, such that for every vertex in another class, the neighborhood of this vertex is a subtree. When the tree is a star (a triad, respectively), the graph is called *star convex bipartite* [8,9] (*triad convex bipartite* [10,9], respectively), where a *triad* is three pathes with a common end. It is known that independent domination is $\mathcal{NP}$-complete for star convex bipartite graphs, but tractable for triad convex bipartite graphs in [14]. In this paper, we simplify the tractability proof in [14].

Our main contributions are making two explicit reductions for independent domination from circular- and triad-convex bipartite graphs respectively to convex bipartite graphs, instead of running modified algorithms such as in [14]. In fact, the second reduction can be viewed as a detailed proof for the correctness of the algorithm in [14], easier to understand with a better modularity. Moreover, our reductions are Cook reductions (i.e. polynomial time Turing reductions) [5], which call the known polynomial time algorithms of independent domination for convex bipartite graphs [3] many times, and also work for *weighted* circular- and

triad-convex bipartite graphs, though the original algorithm in [3] only works for unweighted bipartite graphs. Before our works, only Karp reduction (i.e. polynomial many-one reduction) [5] from circular convex bipartite graphs to circular-arc graphs is used [11]. Thus, our methods may be of use to show more problems tractable for circular- and triad-convex bipartite graphs.

This paper is structured as follows. After introducing necessary definitions and notations mainly from graph theory (Section 2), polynomial time reductions for independent domination from circular-convex bipartite graphs (Section 3) and triad-convex bipartite graphs (Section 4) to convex bipartite graphs are shown respectively. Concluding remarks are at the last section (Section 5).

2 Preliminaries

A *graph* $G = (V, E)$ consists of a vertex set V and an edge set E. Each edge e in E is *incident* to two vertices, called its ends, and these two ends are called *adjacent* to each other. For each vertex v, its *neighborhood* $N(v) = \{u | v \text{ is adjacent to } u\}$, its *closed neighborhood* $N[v] = N(v) \cup \{v\}$. For a subset V' of vertices, $N(V') = \bigcup_{v \in V'} N(v)$. A *path* in a graph is a sequence of different vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$, such that each two consecutive vertices are adjacent to each other. A *cycle* is a path where v_{i_1} and v_{i_k} are also adjacent to each other. A graph is *connected* if every two vertices are connected by a path. A *tree* is a connected graph without any cycle. For a subset V' of vertices, the *induced subgraph* $G[V'] = (V', E')$, where $V' \subseteq V$ and $E' = \{e \in E | e \text{ has both ends in } V'\}$. An *independent set* is a subset of vertices whose induced subgraph has no edge.

In a *weighted* graph $G = (V, E, w)$, there is a function w defined on V, such that each vertex v has a weight $w(v)$. The weight of a vertex subset V' is $w(V') = \sum_{v \in V'} w(v)$. When $w(v) = 1$ for all vertices v, the graph is called *unweighted*. In a *finite* graph, both V and E are finite sets. A *simple* graph has no loop and no parallel edges, where a loop has the same one vertex as its ends, and two parallel edges are incident to the same two ends. In a *bipartite* graph, denoted by $G = (A, B, E)$, the vertex set V is divided into two classes A and B, such that each edge is incident to a vertex in A and a vertex in B respectively. In this paper, we only consider finite simple bipartite graphs.

The *cardinality* of a set X, i.e. the number of elements in X, is denoted by $|X|$. The *difference* of two sets X and Y is denoted by $X \backslash Y = \{x \mid x \in X \text{ and } x \notin Y\}$. The *empty set* is denoted by $\emptyset$. An arbitrary ordering on a set is denoted by $\prec$.

Definition 1. *(*Independent Dominating Set*) In a graph $G = (V, E)$, an independent dominating set D is a subset of V, such that D is an independent set, and for each vertex v in V, either $v \in D$ or $N(v) \cap D \neq \emptyset$.*

Definition 2. *(*Triad*) A $G = (V, E)$ is called a* triad*, if the vertex set V can be partitioned into four parts, $V_1, V_2, V_3, \{v_0\}$, such that for $i = 1, 2, 3$, $V_i \cup \{v_0\}$ induces a path. The vertex v_0 is called* center*.*

Definition 3. *(*Circular Convex Bipartite Graphs *[11]) A bipartite graph $G = (A, B, E)$ is called* circular convex bipartite*, if there is a circular ordering $\prec$*

defined on $A = \{a_1, \ldots, a_n\}$, $a_1 \prec a_2 \prec \cdots \prec a_n \prec a_1$, *such that for each vertex* b *in* B, *its neighborhood* $N(b)$ *is a circular arc under this circular ordering, that is, there are two (possibly equal) vertices* a_i *and* a_j, *where* $1 \leq i \leq j \leq n$, *such that* $N(b) = \{a_i, a_{i+1}, \ldots, a_j\}$ *or* $N(b) = \{a_j, a_{j+1}, \ldots, a_n, a_1, \ldots, a_i\}$.

Definition 4. *(*Triad Convex Bipartite Graphs *[8,9]) A bipartite graph* $G = (A, B, E)$ *is called* triad convex bipartite, *if these is a triad* $T = (A, F)$ *defined on* A, *such that for each vertex* b *in* B, *its neighborhood* $N(b)$ *is a subtree of* T.

Remark 1. The adjacent matrices of circular convex (convex, respectively) bipartite graphs have the so-called *circular* (*consecutive*, respectively) *ones property*, which are recognizable in *linear* time [4]. Tree convex bipartite graphs are also recognizable in *linear* time [1]. The associated circular orderings (trees, respectively) are all constructible in *linear* time, thus can safely be assumed as part of the inputs. Chordal bipartite graphs are recognizable in *square* time.

We refer to [5] for the notions of *polynomial time*, *reductions*, and $\mathcal{NP}$*-completeness*.

3 Reduction from Circular-Convex Bipartite Graphs

In this section, we show that independent domination is polynomial time solvable for circular-convex bipartite graphs, by a polynomial time reduction for this problem from circular-convex bipartite graphs to convex bipartite graphs.

Theorem 1. *For circular convex bipartite graphs* $G = (A, B, E)$ *with a circular ordering on* A, *independent domination is* $O\left(|A|(|A| + |B|)^3\right)$ *time solvable.*

Proof. Without loss of generality, we assume that G contains *no* isolated vertex, since isolated vertices are trivially in every independent dominating set.

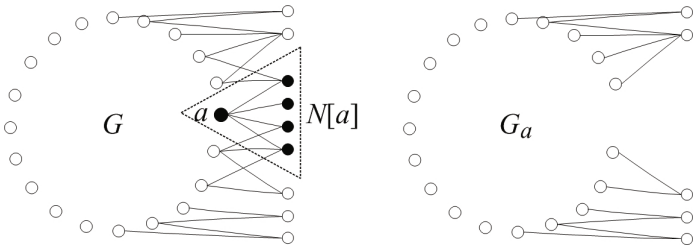


Fig. 2. Removing vertices in $N[a]$ from graph G results in graph G_a

First, for each vertex a in A, we define a graph G_a as follows, see Figure 2.

$$G_a = (A_a, B_a, E_a), \text{ where } A_a = A \setminus \{a\}, B_a = B \setminus N(a), \text{ and}$$
$$E_a = \{e \in E \mid e \text{ is } \textit{not} \text{ incident to any vertex in } N[a]\}.$$

Lemma 1. *For each $a \in A$, G_a is convex bipartite.*

Proof. We prove by definition of convex bipartite graphs. After removing $\{a\} \cup N(a)$ and the incident edges from G, no vertex in $B_a = B \setminus N(a)$ is adjacent to vertex a. Since G is circular convex bipartite, for each vertex in B_a, its neighborhood is a circular arc contained in $A_a = A \setminus \{a\}$. Thus, we can restrict the circular ordering on A to a linear ordering on A_a, such that for each vertex in B_a, its neighborhood is an interval under this linear ordering. □

Lemma 2. *For each $a \in A$, if D is an independent dominating set of G containing a, then $D \setminus \{a\}$ is an independent dominating set of G_a.*

Proof. We prove by definition of independent dominating sets. Since $a \in D$, $N(a) \cap D = \emptyset$. For each vertex $a' \in A_a$, either $a' \in D$ or $N(a') \cap D \neq \emptyset$. Since $a \notin N[a']$, either $a' \in D \setminus \{a\}$ or $N(a') \cap (D \setminus \{a\}) \neq \emptyset$. For each vertex $b' \in B_a$, either $b' \in D$ or $N(b') \cap D \neq \emptyset$. Since $a \notin N[b']$, either $b' \in D \setminus \{a\}$ or $N(b') \cap (D \setminus \{a\}) \neq \emptyset$. □

Lemma 3. *For each $a \in A$, if D' is an independent dominating set of G_a, then $D' \cup \{a\}$ is an independent dominating set of G.*

Proof. We prove again by definition. Since G_a is resulted by removing $N[a]$ from G, a is not adjacent to any vertex in G', $D' \cup \{a\}$ is an independent set. Since D' is an independent dominating set of G_a and each vertex in $N(a)$ is adjacent to a, $D' \cup \{a\}$ is an independent dominating set of G. □

Next, we define a set $\mathcal{S}$ as follows.

$$\mathcal{S} = \{B\} \cup \{D_a \cup \{a\} \mid a \in A \text{ and } D_a \text{ is a minimum independent dominating set in } G_a\}.$$

Remark 2. For each a, G_a is unique, but for each G_a, D_a may not be unique. For our purpose, however, for each a, we only need one such D_a in $\mathcal{S}$, see proof of Lemma 5 below.

Lemma 4. *$\mathcal{S}$ contains a minimum independent dominating set of G.*

Proof. Let D be a minimum independent dominating set of G. We consider the following two cases.

Case 1: $D \cap A = \emptyset$.

Since D is an independent dominating set, for each vertex b in B, either $b \in D$ or $N(b) \cap D \neq \emptyset$. Since $D \cap A = \emptyset$, G is bipartite and $N(b) \subseteq A$, we have $N(b) \cap D = \emptyset$ and thus $b \in D$. So in Case 1 we have $D = B$ and thus $D \in \mathcal{S}$.

Case 2: $D \cap A \neq \emptyset$.

Assume that $a \in D \cap A$. For any minimum independent dominating set D_a of G_a, by Lemma 2, $|D_a| \leq |D| - 1$, and by Lemma 3, $|D| \leq |D_a| + 1$, thus $|D| = |D_a| + 1 = |D_a \cup \{a\}|$. By Lemma 3 and the minimality of D in G, $D_a \cup \{a\}$ is a *minimum* independent dominating set of G. □

Lemma 5. *$\mathcal{S}$ is computable in $O\left(|A|(|A|+|B|)^3\right)$ time.*

Proof. By Lemmm 1, for each $a \in A$, G_a is convex bipartite, thus we can compute a minimum independent dominating set D_a of G_a by the known $O\left((|A|+|B|)^3\right)$ time algorithm in [3]. As remarked in Remark 2, for each a, we only need one such D_a in $\mathcal{S}$. Thus, by an *enumeration* of all $|A|$ vertices a in A, we can compute $\mathcal{S}$ in $O\left(|A|(|A|+|B|)^3\right)$ time. □

Finally, by Lemmas 4 and 5, we can find a minimum independent dominating set of G in $O\left(|A|(|A|+|B|)^3\right)$ time.

This finishes the proof of Theorem 1. □

Remark 3. The above reduction also works for *weighted* independent domination. The only change is in replacing $|D| = |D_a| + 1$ by $w(D) = w(D_a) + w(a)$ in proof of Lemma 4. However, the known polynomial time algorithm in [3] only works for *unweighted* independent domination.

4 Reduction from Triad-Convex Bipartite Graphs

In this section, we show that independent domination is polynomial time solvable for triad-convex bipartite graphs, by a polynomial time reduction for this problem from triad-convex bipartite graphs to convex bipartite graphs. Due to space limitation, we omit some details in this section.

Theorem 2. *For triad convex bipartite graphs $G = (A, B, E)$ with a triad T defined on A, independent domination is $O\left(|A|^3(|A|+|B|)^3\right)$ time solvable.*

Proof. Without loss of generality, we assume that G contains *no* isolated vertex, since isolated vertices are trivially in every independent dominating set.

We assume that A is divided into four parts, $A_1, A_2, A_3, \{a_0\}$, such that for $i = 1, 2, 3$, $A_i \cup \{a_0\}$ induces a path of T. To be specific, we assume that $A_i = \{a_{i,1}, a_{i,2}, \ldots, a_{i,n_i}\}$, where $\sum_{i=1}^{3} n_i = |A| - 1$ and $a_0 a_{i,1} a_{i,2} \cdots a_{i,n_i}$ are three paths of T with a common end a_0.

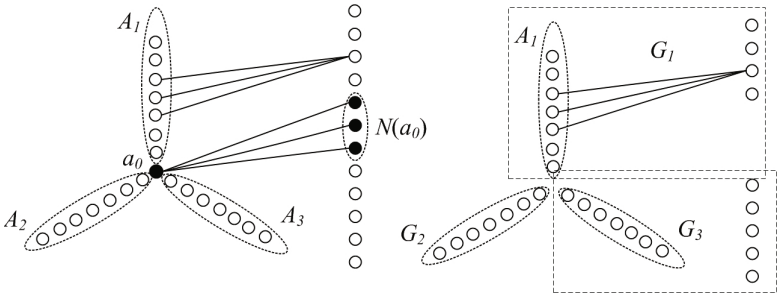


Fig. 3. Removing vertices in $N[a_0]$ from graph G results in three graphs G_1,G_2,G_3

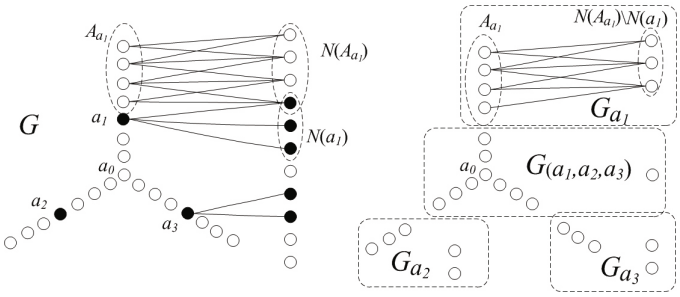


Fig. 4. Removing vertices in $\bigcup_{i=1}^{3}(\{a_i\} \cup N(a_i))$ from graph G results in four graphs G_{a_1},G_{a_2},G_{a_3} and $G_{(a_1,a_2,a_3)}$

First, we define three graphs G_1, G_2, G_3 as follows, see Figure 3.

$$\begin{aligned} G_i &= (A_i, N(A_i) \setminus N(a_0), E_i), \text{ for } i = 1, 2, 3, \text{ where} \\ E_i &= \{e \in E \mid e \text{ is incident to a vertex in } A_i \\ &\qquad \text{but not incident to a vertex in } N(a_0)\}, \end{aligned}$$

For $i = 1, 2, 3$, for each vertex $a_i = a_{i,j_i}$ in A_i, we define four graphs G_{a_1}, G_{a_2}, G_{a_3}, $G_{(a_1,a_2,a_3)}$ as follows, see Figure 4.

$$\begin{aligned} G_{a_i} &= (A_{a_i}, B_{a_i}, E_{a_i}), \text{ where} \\ A_{a_i} &= \{a_{i,j_i+1}, \ldots, a_{i,n_i}\}, B_{a_i} = N(A_{a_i}) \setminus N(a_i), \\ E_{a_i} &= \{e \in E \mid e \text{ is incident to a vertex in } B_{a_i}\}, \text{ and} \end{aligned}$$

$$\begin{aligned} G_{(a_1,a_2,a_3)} &= (A_{(a_1,a_2,a_3)}, B_{(a_1,a_2,a_3)}, E_{(a_1,a_2,a_3)}), \text{ where} \\ A_{(a_1,a_2,a_3)} &= A \setminus \bigcup_{i=1}^{3}(A_{a_i} \cup \{a_i\}), B_{(a_1,a_2,a_3)} = B \setminus \bigcup_{i=1}^{3}(B_{a_i} \cup \{N(a_i)\}), \text{ and} \\ E_{(a_1,a_2,a_3)} &= \{e \in E \mid e \text{ is incident to a vertex in } B_{(a_1,a_2,a_3)}\}. \end{aligned}$$

We define graphs $G_{(a_1,a_2,*)}$ as follows, $G_{(*,a_2,a_3)}, G_{(a_1,*,a_3)}$ are similar.

$$\begin{aligned} G_{(a_1,a_2,*)} &= (A_{(a_1,a_2,*)}, B_{(a_1,a_2,*)}, E_{(a_1,a_2,*)}), \text{ where} \\ A_{(a_1,a_2,*)} &= A \setminus \bigcup_{i\in\{1,2\}} (A_{a_i} \cup \{a_i\}), B_{(a_1,a_2,*)} = B \setminus \left(N(a_i) \bigcup_{i\in\{1,2\}} B_{a_i} \right), \text{ and} \\ E_{(a_1,a_2,*)} &= \{e \in E \mid e \text{ is incident to a vertex in } A_{(a_1,a_2,*)} \\ &\qquad \text{and a vertex in } B_{(a_1,a_2,*)}\}. \end{aligned}$$

We define graphs $G_{(a_1,*,*)}$ as follows, $G_{(*,a_2,*)}, G_{(*,*,a_3)}$ are similar.

$$G_{(a_1,*,*)} = (A_{(a_1,*,*)}, B_{(a_1,*,*)}, E_{(a_1,*,*)}), \text{ where}$$

$$A_{(a_1,*,*)} = A \setminus (A_{a_1} \cup \{a_1\}), B_{(a_1,*,*)} = B \setminus (N(a_1) \cap B_{a_1}), \text{ and}$$
$$E_{(a_1,*,*)} = \{e \in E \mid e \text{ is incident to a vertex in } A_{(a_1,*,*)} \text{ and a vertex in } B_{(a_1,*,*)}\}.$$

Definition 5. *For $i = 1, 2, 3$ and for each $a_i \in A_i$, a triple (a_1, a_2, a_3) is called* good, *if $B_{(a_1,a_2,a_3)}$ is an independent dominating set of $G_{(a_1,a_2,a_3)}$.*
Similarly for $(a_1, a_2, *)$, $(a_1, *, a_3)$,$(*, a_2, a_3)$,$(a_1, *, *)$,$(*, a_2, *)$,$(*, *, a_3)$.

Remark 4. A triple (a_1, a_2, a_3) is *good*, if and only if there is an independent dominating set D of G, such that D contains $\{a_1, a_2, a_3\}$ and a_i is the first vertex of D on the path $a_0 a_{i,1} \cdots a_{i,n_i}$ of the triad T for $i = 1, 2, 3$. Similarly for $(a_1, a_2, *)$, $(a_1, *, a_3)$, $(*, a_2, a_3)$, $(a_1, *, *)$, $(*, a_2, *)$, $(*, *, a_3)$. A star $*$ on the i-th place of a triple means that no vertex in $\{a_0\} \cup A_i$ is in D for $i = 1, 2, 3$.

Lemma 6. *For $i = 1, 2, 3$ and for each $a_i \in A_i$, G_i and G_{a_i} are convex bipartite.*

Proof. We prove by definition of convex bipartite graphs for G_{a_i}, the proof for G_i is similar and thus omitted. After removing $N(a_i)$ and the incident edges from G, no vertex in $B_{a_i} = N(A_{a_i}) \setminus N(a_i)$ is adjacent to vertex a_i. Since G is triad convex bipartite, for each vertex in B_{a_i}, its neighborhood is a path of T on $A_{a_i} = \{a_{i,j_i+1}, \ldots, a_{i,j_n}\}$. Thus, we can define a linear ordering $\prec_i$ on A_{a_i}, $a_{i,j_i+1} \prec_i \cdots \prec_i a_{i,j_n}$, such that for each vertex in B_{a_i}, its neighborhood is an interval under this linear ordering $\prec_i$. □

Lemma 7. *For each triple (a_1, a_2, a_3), if D is an independent dominating set of G containing a_i for $i = 1, 2, 3$, then $D \cap (A_{a_i} \cup B_{a_i})$ is an independent dominating set of G_{a_i} for $i = 1, 2, 3$. Similarly for $(a_1, a_2, *)$, $(a_1, *, a_3)$, $(*, a_2, a_3)$, $(a_1, *, *)$, $(*, a_2, *)$, $(*, *, a_3)$.*

Proof. We prove by definition of independent dominating sets. Since $a_i \in D$, $N(a_i) \cap D = \emptyset$. For each vertex $a' \in A_{a_i}$, either $a' \in D$ or $N(a') \cap D \neq \emptyset$. Since $a_i \notin \{a'\} \cup N(a')$, either $a' \in D \cap A_{a_i}$ or $N(a') \cap (D \cap B_{a_i}) \neq \emptyset$. For each vertex $b' \in B_{a_i}$, either $b' \in D$ or $N(b') \cap D \neq \emptyset$. Since $a_i \notin \{b'\} \cup N(b')$, either $b' \in D \cap B_{a_i}$ or $N(b') \cap (D \cap A_{a_i}) \neq \emptyset$. Similarly for $(a_1, a_2, *)$, $(a_1, *, a_3)$, $(*, a_2, a_3)$, $(a_1, *, *)$, $(*, a_2, *)$, $(*, *, a_3)$. □

Lemma 8. *For each good triple (a_1, a_2, a_3), if D_{a_i} is an independent dominating set of G_{a_i} for $i = 1, 2, 3$, then $B_{(a_1,a_2,a_3)} \cup \bigcup_{i=1}^{3}(D_{a_i} \cup \{a_i\})$ is an independent dominating set of G. Similarly for $(a_1, a_2, *)$, $(a_1, *, a_3)$, $(*, a_2, a_3)$, $(a_1, *, *)$, $(*, a_2, *)$, $(*, *, a_3)$.*

Proof. We prove by definition. Since for $i = 1, 2, 3$, G_{a_i} is resulted by removing $\{a_i\} \cup N(a_i)$ from G, a_i is not adjacent to any vertex in G_{a_i} and $G_{(a_1,a_2,a_3)}$, $B_{(a_1,a_2,a_3)} \cup \bigcup_{i=1}^{3}(D_{a_i} \cup \{a_i\})$ is an independent set. Since the triple (a_1, a_2, a_3) is good, B_{a_1,a_2,a_3} is an independent dominating set of $G_{(a_1,a_2,a_3)}$. Since D_{a_i} is an independent dominating set of G_{a_i} and each vertex in $N(a_i)$ is adjacent to a_i, $B_{(a_1,a_2,a_3)} \cup \bigcup_{i=1}^{3}(D_{a_i} \cup \{a_i\})$ is an independent dominating set of G. Similarly for $(a_1, a_2, *)$, $(a_1, *, a_3)$, $(*, a_2, a_3)$, $(a_1, *, *)$, $(*, a_2, *)$, $(*, *, a_3)$. □

Next, we define a set $\mathcal{S} = \{B, \{a_0\} \cup D_1 \cup D_2 \cup D_3\} \cup \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3$ as follows.

$$\mathcal{S}_1 = \{B_{(a_1,*,*)} \cup D_{a_1} \cup \{a_1\}, B_{(*,a_2,*)} \cup D_{a_2} \cup \{a_2\}, B_{(*,*,a_3)} \cup D_{a_3} \cup \{a_3\} \mid a_i \in A_i \text{ and } D_{a_i} \text{ is a minimum independent dominating set in } G_{a_i} \text{ for } i = 1,2,3\},$$

$$\mathcal{S}_2 = \{B_{(a_1,a_2,*)} \cup \bigcup_{i\in\{1,2\}} (D_{a_i} \cup \{a_i\}), B_{(*,a_2,a_3)} \cup \bigcup_{i\in\{2,3\}} (D_{a_i} \cup \{a_i\}), B_{(a_1,*,a_3)} \cup \bigcup_{i\in\{1,3\}} (D_{a_i} \cup \{a_i\}) \mid a_i \in A_i \text{ and } D_{a_i} \text{ is a minimum independent dominating set in } G_{a_i} \text{ for } i = 1,2,3\},$$

$$\mathcal{S}_3 = \{B_{(a_1,a_2,a_3)} \cup \bigcup_{i\in\{1,2,3\}} (D_{a_i} \cup \{a_i\}) \mid (a_1,a_2,a_3) \text{ is good and } D_{a_i} \text{ is a minimum independent dominating set in } G_{a_i} \text{ for } i = 1,2,3\},$$

where for $i = 1, 2, 3$, D_i is a minimum dominating set of G_i and G_i is resulted by removing $N[a_0]$ from G.

Remark 5. For each triple (a_1, a_2, a_3), G_{a_i} is unique, but for each G_{a_i}, D_{a_i} may not be unique. For our purpose, however, for each (a_1, a_2, a_3), we only need one triple $(D_{a_1}, D_{a_2}, D_{a_3})$ in $\mathcal{S}$, see proof of Lemma 10 below. Similarly for $(a_1, a_2, *)$, $(a_1, *, a_3)$, $(*, a_2, a_3)$, $(a_1, *, *)$, $(*, a_2, *)$, $(*, *, a_3)$.

Lemma 9. *$\mathcal{S}$ contains a minimum independent dominating set of G.*

Proof. Let D be a minimum independent dominating set of G. We consider the following *five* cases.

Case 1: $D \cap A = \emptyset$. In this case we have $D = B$, which is in $\mathcal{S}$.

Case 2: $a_0 \in D$. In this case, similar to the reasoning process in Case 5 below, we have $|D| = |\{a_0\} \cup \bigcup_{i\in\{1,2,3\}} D_i|$, thus $\{a_0\} \cup \bigcup_{i\in\{1,2,3\}} D_i$ is a minimum independent dominating set of G, which is in $\mathcal{S}$.

Case 3: $D \cap A_i \neq \emptyset$ for some $i \in \{1, 2, 3\}$ but $D \cap (\{a_0\} \cup A_j) = \emptyset$ for $j \neq i$. In this case, similar to Case 5 below, a minimum dominating set of G is in $\mathcal{S}_1$.

Case 4: $D \cap (\{a_0\} \cup A_i) = \emptyset$ for some $i \in \{1, 2, 3\}$ but $D \cap A_j \neq \emptyset$ for $j \neq i$. In this case, similar to Case 5 below, a minimum dominating set of G is in $\mathcal{S}_2$.

Case 5: $D \cap A_i \neq \emptyset$ for $i = 1, 2, 3$. Assume that $a_i \in D \cap A_i$ for $i = 1, 2, 3$ and the triple (a_1, a_2, a_3) is good. For any minimum independent dominating sets D_{a_i} of G_{a_i} for $i = 1, 2, 3$, by Lemma 7, $\sum_{i=1}^{3} |D_{a_i}| \leq |D| - |B_{(a_1,a_2,a_3)}| - 3$, and by Lemma 8, $|D| \leq \sum_{i=1}^{3} |D_{a_i}| + |B_{(a_1,a_2,a_3)}| + 3$, thus $|D| = \sum_{i=1}^{3} |D_{a_i}| + |B_{(a_1,a_2,a_3)}| + 3 = |B_{(a_1,a_2,a_3)} \cup \bigcup_{i=1}^{3} (D_{a_i} \cup \{a_i\})|$. By Lemma 8 and the minimality of D in G, $B_{(a_1,a_2,a_3)} \cup \bigcup_{i\in\{1,2,3\}} (D_{a_i} \cup \{a_i\})$ is a *minimum* independent dominating set of G, which is in $\mathcal{S}_3$. □

Lemma 10. *$\mathcal{S}$ is computable in $O\left(|A|^3(|A| + |B|)^3\right)$ time.*

Proof. By Lemmm 6, for $i = 1, 2, 3$ and for each $a_i \in A_i$, G_{a_i} is convex bipartite, thus we can compute a minimum independent dominating set D_{a_i} of G_{a_i} by the

known $O\left((|A|+|B|)^3\right)$ time algorithm in [3]. As remarked in Remark 5, for each good triple (a_1, a_2, a_3), we only need one such triple $(D_{a_1}, D_{a_2}, D_{a_3})$ in $\mathcal{S}$. Thus, by an *enumeration* of all $|A_1||A_2||A_3|$ triples (a_1, a_2, a_3), we can compute $\mathcal{S}_3$ in $O\left(|A|^3(|A|+|B|)^3\right)$ time. Similarly for $\mathcal{S}_1$ and $\mathcal{S}_2$. □

Finally, by Lemmas 9 and 10, we can find a minimum independent dominating set of G in $O\left(|A|^3(|A|+|B|)^3\right)$ time.

This finishes the proof of Theorem 2. □

Remark 6. The above reduction also works for *weighted* independent domination. The only changes are in replacing $|D| = \sum_{i=1}^{3} |D_{a_i}| + |B_{(a_1,a_2,a_3)}| + 3$ by $w(D) = \sum_{i=1}^{3}(w(D_{a_i}) + w(a_i)) + w(B_{(a_1,a_2,a_3)})$ and so on in proof of Lemma 9.

5 Concluding Remarks

We have shown that independent domination is polynomial time reducible from circular- and triad-convex bipartite graphs to convex bipartite graphs. As in [12], we make Cook reductions from circular convex bipartite graphs to convex bipartite graphs. Our methods may be of use to show more problems tractable for circular- and triad-convex bipartite graphs. It would be interesting to find real applications of these results.

Recently, maximum non-crossing matching for convex bipartite graphs is studied [2]. Whether the results in [2] carry over for circular- and triad-convex bipartite graphs is still unknown.

Acknowledgments. We thank Professor Kaile Su for his encouragements and supports to this work. We thank Professor Francis Y.L. Chin for bringing our attention to the notion of circular convex bipartite graphs during FAW-AAIM 2011. We also thank the unknown reviewers of FAW-AAIM 2013 for helpful comments.

References

1. Bao, F.S., Zhang, Y.: A review of tree convex sets test. Comput. Intell. 28(3), 358–372 (2012), Old version: A survey of tree convex sets test. arXiv.0906.0205 (2009)
2. Chen, D.Z., Liu, X., Wang, H.: Computing maximum non-crossing matching in convex bipartite graphs. In: Snoeyink, J., Lu, P., Su, K., Wang, L. (eds.) FAW-AAIM 2012. LNCS, vol. 7285, pp. 105–116. Springer, Heidelberg (2012)
3. Damaschke, P., Müller, H., Kratsch, D.: Domination in convex and chordal bipartite graphs. Inf. Process. Lett. 36(5), 231–236 (1990)
4. Dom, M.: Algorithmic aspects of the consecutive ones property. Bulletin of the EATCS 98, 27–59 (2009)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company (1979)

6. Golumbic, M.C., Goss, C.F.: Perfect elimination and chordal bipartite graphs. J. Graph Theory 2, 155–163 (1978)
7. Grover, F.: Maximum matching in a convex bipartite graph. Nav. Res. Logist. Q. 14, 313–316 (1967)
8. Jiang, W., Liu, T., Ren, T., Xu, K.: Two hardness results on feedback vertex sets. In: Atallah, M., Li, X.-Y., Zhu, B. (eds.) FAW-AAIM 2011. LNCS, vol. 6681, pp. 233–243. Springer, Heidelberg (2011)
9. Jiang, W., Liu, T., Wang, C., Xu, K.: Feedback vertex sets on restricted bipartite graphs. Theor. Comput. Sci (in press, 2013), doi:10.1016/j.tcs.2012.12.021
10. Jiang, W., Liu, T., Xu, K.: Tractable feedback vertex sets in restricted bipartite graphs. In: Wang, W., Zhu, X., Du, D.-Z. (eds.) COCOA 2011. LNCS, vol. 6831, pp. 424–434. Springer, Heidelberg (2011)
11. Liang, Y.D., Blum, N.: Circular convex bipartite graphs: maximum matching and Hamiltonian circuits. Inf. Process. Lett. 56, 215–219 (1995)
12. Lu, Z., Liu, T., Xu, K.: Tractable connected domination for restricted bipartite graphs. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 721–728. Springer, Heidelberg (2013)
13. Müller, H., Brandstät, A.: The NP-completeness of steiner tree and dominating set for chordal bipartite graphs. Theor. Comput. Sci. 53(2-3), 257–265 (1987)
14. Song, Y., Liu, T., Xu, K.: Independent domination on tree convex bipartite graphs. In: Snoeyink, J., Lu, P., Su, K., Wang, L. (eds.) FAW-AAIM 2012. LNCS, vol. 7285, pp. 129–138. Springer, Heidelberg (2012)
15. Wang, C., Liu, T., Jiang, W., Xu, K.: Feedback vertex sets on tree convex bipartite graphs. In: Lin, G. (ed.) COCOA 2012. LNCS, vol. 7402, pp. 95–102. Springer, Heidelberg (2012)

Spanning Distribution Trees of Graphs
(Extended Abstract)

Masaki Kawabata and Takao Nishizeki

School of Science and Technology, Kwansei Gakuin University,
2-1 Gakuen, Sanda, 669-1337 Japan
masaki.kawabata.kwansei@gmail.com, nishi@kwansei.ac.jp

Abstract. Let G be a graph with a single source w, assigned a positive integer called the supply. Every vertex other than w is a sink, assigned a nonnegative integer called the demand. Every edge is assigned a positive integer called the capacity. Then a spanning tree T of G is called a spanning distribution tree if the capacity constraint holds when, for every sink v, an amount of flow, equal to the demand of v, is sent from w to v along the path in T between them. The spanning distribution tree problem asks whether a given graph has a spanning distribution tree or not. In the paper, we first observe that the problem is NP-complete even for series-parallel graphs, and then give a pseudo-polynomial time algorithm to solve the problem for a given series-parallel graph G.

Keywords: spanning distribution tree, series-parallel graph, flow, supply, demand, partial k-tree.

1 Introduction

Let G be a graph with a single *source* w, which is assigned a positive integer $\sup(w)$, called the *supply* of w. One may assume without loss of generality that every vertex v other than the source w is a *sink* and is assigned a nonnegative integer $\mathrm{dem}(v)$, called the *demand* of v. Every edge e of G is assinged a positive integer $\mathrm{cap}(e)$, called the *capacity* of e. Figure 1 illustrates such a graph, in which the source w is drawn as a square, a sink is drawn as a circle, the integer in a square or circle is a supply or demand, and the integer attached to an edge is the capacity.

As in an ordinary flow network, the source w can send at most an amount $\sup(w)$ of flow to sinks through edges in G, every sink v must receive an amount $\mathrm{dem}(v)$ of flow from w, and hence $\sup(w)$ must be at least the sum D of all demands in G. However, we wish to let the flow to v run along the path from w to v in the same spanning tree T of G for every sink v; T is drawn by thick lines in Fig. 1. Of course, the *capacity constraint* must be satisfied for every edge e of T: the amount of flow through e should not exceed the capacity $\mathrm{cap}(e)$ of e. Regard T as a tree rooted at the source w as illustrated in Fig. 1, then the amount of flow through edge $e = (u, u')$ of T is equal to the sum of demands of u' and its all descendants if u is the parent of u' in the rooted tree T. In Fig. 1, the integer

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 153–162, 2013.

in parentheses attached to an edge e in T is the amount of flow through e. We call such a spanning tree T a *spanning distribution tree* of G. The *spanning distribution tree problem* asks whether there exists a spanning distribution tree in a given graph G. The problem has some applications to the power supply problem for power delivery networks [1, 5–8, 11, 14], the server-client problem in computer networks [9], etc.

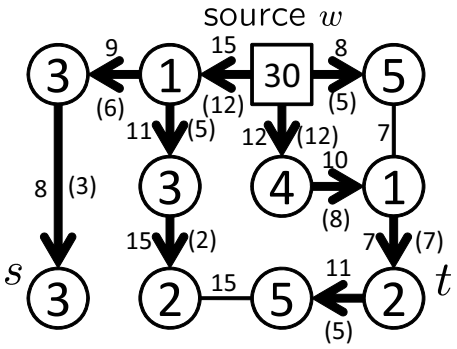


Fig. 1. Series-parallel graph G and its spanning distribution tree

The spanning distribution tree problem looks like an ordinary network flow problem with a single source and multiple sinks. However, in our case, the flow to every sink v must be a single "path flow." In this sense, our problem looks like the "unsplittable flow problem"[2, 3, 10]. However, in our case, the flow paths for all sinks must induce a spanning tree $T$of G.

A *spanning distribution forest* and the *spanning distribution forest problem* are similarly defined for a graph G with two or more sources. There are linear-time algorithms to solve the spanning distribution forest problem when G is a tree without capacity [7] or with capacity [8]. Thus, it is desired to obtain an efficient algorithm to solve the spanning distribution tree (or forest) problem for a larger class of graphs, say series-parallel graphs. Many problems including the Steiner tree problem can be solved in linear time for series-parallel graphs [13], although there are a few problems, including the edge-disjoint paths problem [12], which are NP-complete for series-parallel graphs.

In this paper, we first observe that the spanning distribution tree problem is NP-complete even for series-parallel graphs. We then give a pseudo-polynomial time algorithm for the problem on a series-parallel graph G. The computation time is bounded by a polynomial in n and D, where n is the number of vertices of G. The algorithm runs in linear time if $D = O(1)$, and runs in polynomial time if D is bounded by a polynomial in n. We finally remark on the spanning distribution forest problem for series-parallel graphs and partial k-trees, that is, graphs of bounded tree-width.

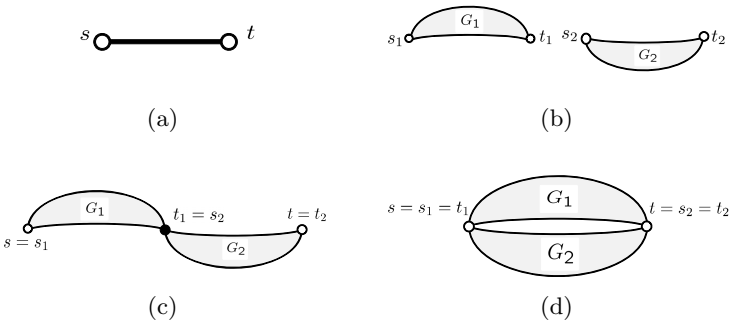


Fig. 2. Illustration for the definition of series-parallel graphs

2 NP-Completeness

A *series-parallel graph* is defined recursively as follows:

1. A graph G of a single edge (s, t), depicted in Fig. 2(a), is a series-parallel graph. The vertices s and t are called the *terminals* of G.
2. Let G_1 be a series-parallel graph with terminals s_1 and t_1, and let G_2 be a series-parallel graph with terminals s_2 and t_2, as depicted in Fig. 2(b).
   (a) A graph G obtained from G_1 and G_2 by identifying vertex t_1 with s_2, as illustrated in Fig. 2(c), is a series-parallel graph, whose terminals are $s = s_1$ and $t = t_2$. Such a connection is called a *series connection*.
   (b) A graph G obtained from G_1 and G_2 by identifying vertex s_1 with s_2 and t_1 with t_2, as illustrated in Fig. 2(d), is a series-parallel graph, whose terminals are $s = s_1 = s_2$ and $t = t_1 = t_2$. Such a connection is called a *parallel connection*.

The graphs in Figs. 1 and 3 are series-parallel graphs.

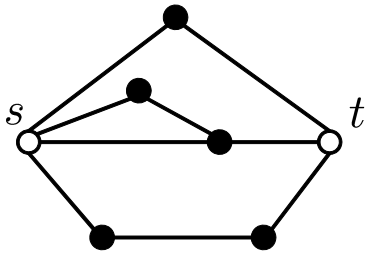


Fig. 3. Series-parallel graph

One can observe that the spanning distribution tree problem is NP-complete even for series-parallel graphs, as follows. Clearly the problem belongs to the class of NP. Therefore, it suffices to show that the set partition problem, which is

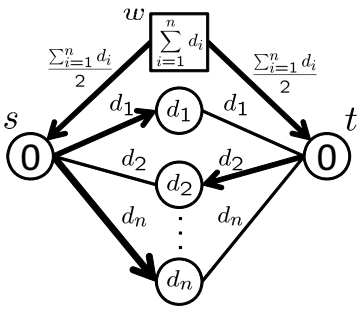


Fig. 4. Series-parallel graph constructed from an instance of a set partition problem

known NP-complete [4, p.47], can be reduced to our problem on a series-parallel graph G in polynomial time. Given a set A of n positive integers $d_1, d_2, \cdots, d_n$, the *set partition problem* asks whether A can be partitioned to two subsets A_1 and A_2 such that the sum of integers in A_1 equals to the sum of integers in A_2. One may assume that $\sum_{i=1}^{n} d_i$ is an even number. We construct a series-parallel graph G as illustrated in Fig. 4. G has a single source w with supply $\sum_{i=1}^{n} d_i$, two sinks s and t with demands 0, and n sinks, each with demand d_i, $1 \leq i \leq n$. The source w is joined to s and t by edges with capacities $\sum_{i=1}^{n} d_i/2$. Each sink with demand d_i is joined to s and t by edges with capacities d_i. The resulting graph G is a series-parallel graph. Clearly, set A has a desired partition if and only if G has a spanning distribution tree. Thus we have the following theorem.

Theorem 1. *The spanning distribution tree problem is NP-complete even for series-parallel graphs.*

Thus, the spanning distribution problem cannot be solved for series-parallel graphs in polynomial time unless P=NP.

3 Pseudo-polynomial Algorithm

In this section we give a pseudo-polynomial time algorithm for the spanning distribution tree problem on a series-parallel graph G.

3.1 Outline of Algorithm

A series-parallel graph G can be represented by a *binary decomposition tree* T_{BD}. Every node u of T_{BD} corresponds to a subgraph G_u of G. If u is a leaf of T_{BD}, then G_u consists of a single edge. Every inner node u of T_{BD} is labeled by **s** or **p**, which represents a series or parallel connection. Let u_1 and u_2 be the children of u in T_{BD}, let G_1 be the graph corresponding to u_1, and let G_2 be the graph corresponding to u_2. Then G_u is a series (or parallel) connection of G_1 and G_2 if u is labeled by **s** (or **p**). T_{BD} can be obtained from G in linear time [13].

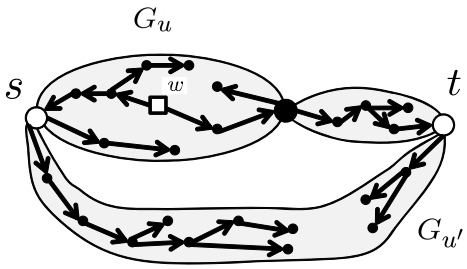


Fig. 5. Illustration of a spanning distribution tree in a series-parallel graph

Our algorithm is based on a dynamic programming approach with the aid of T_{BD}. Consider a spanning distribution tree T of a series-parallel graph G. T is drawn by thick lines in Fig. 5. Let u and u' be inner nodes of the binary decomposition tree T_{BD} of G, and let G_u and $G_{u'}$ be the subgraphs corresponding to u and u', respectively, as illustrated in Fig. 5. T induces forests F_u and $F_{u'}$ in G_u and $G_{u'}$, respectively. In this example, the source w is contained in G_u, and the forest F_u is a spanning distribution tree of G_u, while the forest $F_{u'}$ is a spanning 2-tree of $G_{u'}$, consisting of two vertex-disjoint trees. Our algorithm finds these spanning trees and 2-trees from leafs to the root of T_{BD}.

A spanning forest F of a graph G with exactly two sources w_1 and w_2 is called a *spanning distribution* 2*-tree* if F consists of two vertex-disjoint trees T_1 and T_2 and the following (a) and (b) hold for $i = 1$ and 2:

(a) w_i is contained in T_i, and the supply $\sup(w_i)$ of w_i is at least the sum of demands in T_i; and
(b) the amount of flow through every edge e in T_i does not exceed the capacity $\mathrm{cap}(e)$ of e if, for each sink v in T_i, an amount $\mathrm{dem}(v)$ of flow is sent from w_i to v along the path in T_i between them.

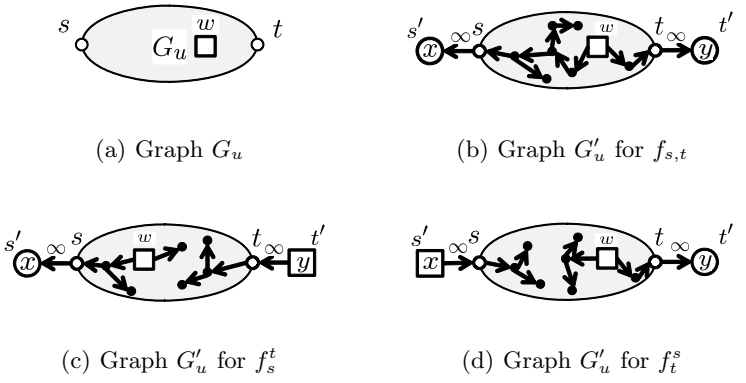
Instead of finding these spanning trees and 2-trees in G_u, we actually compute six functions $f_{s,t}$, f_s^t, f_t^s, g_t^s, g_s^t and $g^{s,t}$, which represent the existence of these trees and 2-trees in G_u. If G_u contains the source w, then a spanning distribution tree or 2-tree may be able to output some amount of flow from the terminals s and t. Otherwise, a tree or 2-tree must be inputted some amount of flow from the terminals. The variables x and y of the six functions represent the amount of flow which can be outputted from G or must be inputted to G_u through terminals s and t, respectively. The subscript s or t of functions f's and g's represents the output terminal of G_u, while the superscript s or t represents the input terminal.

3.2 Definitions of Functions

One may assume that the sum D of all demands in G does not exceed $\sup(w)$; otherwise, there is no spanning distribution tree in G. Of course, the amount of flow through an edge does not exceed D. Denote by $\mathbb{Z}_D$ the set of all integer z such that $0 \leq z \leq D$. Thus, the variables x and y run over $\mathbb{Z}_D$. The range of

the six functions is $\{0, 1\}$; "1" means that G_u has a spanning distribution tree or 2-tree with some amount of input or output flow specified by x and y, while "0" means that G_u does not have such a tree or 2-tree. The three functions $f_{s,t}$, f_s^t and f_t^s are defined only for a subgraph G_u containing the source w, while the other three function g_t^s, g_s^t and $g^{s,t}$ are defined only for a subgraph G_u which does not contain w.

The six functions formally defined for $x, y \in \mathbb{Z}_D$, as follows.



(a) Graph G_u

(b) Graph G'_u for $f_{s,t}$

(c) Graph G'_u for f_s^t

(d) Graph G'_u for f_t^s

Fig. 6. Illustration for $f_{s,t}$, f_s^t and f_t^s

(i) $f_{s,t}(G_u, x, y)$

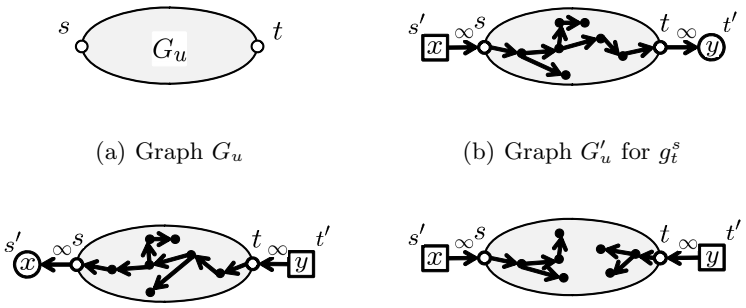
G_u contains w as illustrated in Fig. 6(a). Let s and t be the terminals of G_u. Add to G_u virtual sinks s' and t', and let $\mathrm{dem}(s') = x$ and $\mathrm{dem}(t') = y$. Add to G_u virtual edges (s, s') and (t, t') with infinite capacities. Let G'_u be the resulting graph. Then $f_{s,t}$ is defined as follows: $f_{s,t}(G_u, x, y) = 1$ if G'_u has a spanning distribution tree. Intuitively speaking, $f_{s,t}(G_u, x, y) = 1$ means that G_u has a spanning distribution tree which can output x and y units of flow from terminals s and t, respectively. For the sake of convenience, we assume that $f_{s,t}(G_u, x, y) = 0$ for all integers $x, y \notin \mathbb{Z}_D$. The same assumption applies to the other five functions.

(ii) $f_s^t(G_u, x, y)$

Add to G_u virtual sinks s' with $\mathrm{dem}(s') = x$ and a virtual source t' with $\sup(t') = y$, and add two virtual edges (s, s') and (t, t') with infinite capacities. Let G'_u be the resulting graph with two sources w and t'. Then the function f_s^t is defined as follows: $f_s^t(G_u, x, y) = 1$ if G'_u has a spanning distribution 2-tree consisting of two trees T_1 and T_2 such that T_1 contains w, s and s' and T_2 contains t' and t. Intuitively speaking, $f_s^t(G_u, x, y) = 1$ means that G_u has a spanning distribution 2-tree which can output x units of flow from s if y units is inputted from t.

(iii) $f_t^s(G_u, x, y)$

The function f_t^s is similarly defined as f_s^t although the roles of s and t are interchanged, as illustrated in Fig. 6(d).



(a) Graph G_u (b) Graph G'_u for g_t^s

(c) Graph G'_u for g_s^t (d) Graph G'_u for $g^{s,t}$

Fig. 7. Illustration for g_t^s, g_s^t, $g^{s,t}$

(iv) $g_t^s(G_u, x, y)$

G_u does not contains the source w as illustrated in Fig. 7(a). Add to G_u a source s' with $\sup(s') = x$ and a sink t' with $\mathrm{dem}(t') = y$. Add to G_u two edges (s, s') and (t, t') with infinite capacities. Let G'_u be the resulting graph. Then $g_t^s(G_u, x, y) = 1$ if G'_u has a spanning distribution tree.

(v) $g_s^t(G_u, x, y)$

The function g_s^t is similarly defined as g_t^s although the roles of s and t are interchanged, as illustrated in Fig. 7(c).

(vi) $g^{s,t}(G_u, x, y)$

Add to G_u two sources s' and t' with $\sup(s') = x$ and $\sup(t') = y$, and add to G_u two edges (s, s') and (t, t') with infinite capacities. Let G'_u be the resulting graph. Then $g^{s,t}(G_u, x, y) = 1$ if G'_u has a spanning distribution 2-tree consisting of two trees T_1 and T_2 such that T_1 contains s' and s and T_2 contains t' and t.

3.3 Algorithm

The terminal s or t of G_u may be the source w or a sink of positive demand. However, we consider a virtual graph G_u^* in which both s and t are regarded as sinks of zero demand, and compute the six functions for G_u^* in place of G_u. Such a convention makes the description of our algorithm simple. One can easily compute the functions for G_u from those for G_u^*.

(a) How to decide the existence of a spanning distribution tree in G

Let r be the root of the decomposition tree T_{BD} of a given series-parallel graph G, then $G = G_r$. Suppose that the functions have been computed for $G^* = G_r^*$. When both of the terminals s and t of G are sinks, G has a spanning distribution tree if and only if $f_{s,t}(G^*, \mathrm{dem}(s), \mathrm{dem}(t)) = 1$. When either s or t, say s, is the source w, G has a spanning distribution tree if and only if $g_t^s(G^*, \sup(w), \mathrm{dem}(t)) = 1$.

(b) Computation at a leaf u of T_{BD}

Let u be a leaf of T_{BD}, then G_u consists of a single edge $e = (s, t)$ as illustrated in Fig. 2(a). G_u^* contains no source, and hence $f_{s,t}$, f_s^t and f_t^s are not defined for G_u^*. Since $\mathrm{dem}(s)$=$\mathrm{dem}(t)$=0 in G_u^*, we can compute g_t^s, g_s^t and $g^{s,t}$ for $x, y \in \mathbb{Z}_D$ as follows:

$$g_t^s(G_u^*, x, y) = \begin{cases} 1 & \text{if } y \leq x \text{ and } y \leq \mathrm{cap}(e); \\ 0 & \text{otherwise;} \end{cases}$$

$$g_s^t(G_u^*, x, y) = \begin{cases} 1 & \text{if } x \leq y \text{ and } x \leq \mathrm{cap}(e); \\ 0 & \text{otherwise;} \end{cases}$$

and

$$g^{s,t}(G_u^*, x, y) = 1.$$

(c) Computation at an inner node u of T_{BD}

Let u be an inner node of T_{BD}, let u_1 and u_2 be the children of u, and let G_1 and G_2 be the subgraphs of G corresponding to u_1 and u_2, respectively. Then G is a series or parallel connection of G_1 and G_2 as illustrated in Figs. 2(c) and (d). One can compute the fix functions of G from those of G_1 and G_2. The details are omitted in this extended abstract.

3.4 Computation Time

The functions g_t^s, g_s^t and $g^{s,t}$ can be computed for a leaf u of T_{BD} in time $O(|\mathbb{Z}_D|^2) = O(D^2)$ as in Subsection 3.3(b). Let n be the number of vertices in a given series-parallel graph G. One may assume that G is a simple graph and has no multiple edges. Then G contains at most $2n-3$ edges, and hence T_{BD} has at most $2n-3$ leaves. Thus, the computation at all the leaves takes time $O(D^2 n)$.

If an inner node u of T_{BD} corresponds to a series connection, then the six functions can be computed in time $O(D^3)$. On the other hand, if u corresponds to a parallel connection, then the six functions can be computed in time $O(D^4)$. Since the binary tree T_{BD} has at most $2n-3$ leaves, T_{BD} has at most $2n-4$ inner nodes. Thus, the computation at all inner nodes takes time $O(D^4 n)$.

As shown in Subsection 3.3(a), one can decide from $f_{s,t}$, g_t^s and g_s^t in time $O(1)$ whether G has a spanning distribution tree.

Thus we have the following theorem.

Theorem 2. *The spanning distribution tree problem can be solved in time* $O(D^4n)$ *for series-parallel graphs.*

4 Concluding Remarks

In the paper, we first formalized the spanning distribution tree problem, and observed that the problem is NP-complete even for series-parallel graphs. We then gave a pseudo-polynomial time algorithm to solve the problem for a series-parallel graph G in time $O(D^4n)$, where n is number of vertices in G and D is the sum of all demands in G. If D is bounded by a polynomial in n, then the algorithm take polynomial time. If D is $O(1)$, then it takes linear time. The algorithm can be modified so that it actually finds a spanning distribution tree whenever G has it.

The time complexity $O(D^4n)$ above can be improved to $O(D^2n)$. In place of the functions f's and g's of two variables $x, y \in \mathbb{Z}_D$ with range {0,1}, consider functions f's and g's of a single variable $x \in \mathbb{Z}_D$ with range $\mathbb{Z}_D$; for example, define $f_{s,t}(G_u, x) = \max\{y \in \mathbb{Z}_D | f_{s,t}(G_u, x, y) = 1\}$. Computing these new functions f's and g's, one can solve the spanning distribution tree problem on series-parallel graphs in time $O(D^2n)$.

Extending our algorithm, one can solve a spanning distribution forest problem for a series-parallel graph containing two or more sources. Furthermore, generalizing our algorithm, one can solve the spanning distribution tree or forest problem for partial k-trees, that is, graphs of bounded tree-width.

If a given graph has no spanning distribution tree, then one wishes to find the *maximum distribution tree*, that is, a distribution tree with the maximum sum of demands. Such a *maximum distribution tree problem* is, of course, NP-hard for series-parallel graphs. The *maximum distribution forest problem* is similarly defined for graphs with two or more sources. Extending our algorithm, one can find the maximum distribution forest in pseudo-polynomial time for series-parallel graphs or partial k-trees. There are fully polynomial-time approximation schemes (FPTASs) for the maximum distribution forest problem when a given graph G is a tree without edge-capacity [7] or with edge-capacity [8], and there is an FPTAS for the maximum distribution tree problem when G is a series-parallel graph without edge-capacity [5]. Thus, it is desired to obtain an FPTAS for the maximum distribution forest problem for series-parallel graphs or partial k-trees with edge-capacity.

Acknowledgments. We thank Kouji Imagawa for his comment on the NP-completeness proof. This research was partially supported by MEXT-Supported Program for the Strategic Research Foundation at Private Universities.

References

1. Boulaxis, N.G., Papadopoulos, M.P.: Optimal feeder routing in distribution system planning using dynamic programming technique and GIS facilities. IEEE Trans. on Power Delivery 17(1), 242–247 (2002)

2. Chekuri, C., Ene, A., Korula, N.: Unsplittable flow in paths and trees and column-restricted packing integer programs. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX-RANDOM 2009. LNCS, vol. 5687, pp. 42–55. Springer, Heidelberg (2009)
3. Dinitz, Y., Garg, N., Goemans, M.X.: On the single-source unsplittable flow problem. Combinatorica 19(1), 17–41 (1999)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness (Twenty-second printing), pp. 90–91. W.H. Freeman and Company (2000)
5. Ito, T., Demaine, E.D., Zhou, X., Nishizeki, T.: Approximability of partitioning graphs with supply and demand. Journal of Discrete Algorithms 6, 627–650 (2008)
6. Ito, T., Zhou, X., Nishizeki, T.: Partitioning graphs of supply and demand. Discrete Applied Math. 157, 2620–2633 (2009)
7. Ito, T., Zhou, X., Nishizeki, T.: Partitioning trees of supply and demand. IJFCS 16(4), 803–827 (2005)
8. Kawabata, M., Nishizeki, T.: Partitioning trees with supply, demand and edge-capacity. In: Proc. of ISORA 2011. Lecture Notes in Operation Research, vol. 14, pp. 51–58 (2011); Also IEICE Trans. on Fundamentals of Electronics, Communications and Computer Science (to appear)
9. Kim, M.S., Lam, S.S., Lee, D.-Y.: Optimal distribution tree for internet streaming media. In: Proc. 23rd Int. Conf. on Distributed Computing System (ICDCS 2003), pp. 116–125 (2003)
10. Kleinberg, J.M.: Single-source unsplittable flow. In: Proc. of 37th FOCS, pp. 68–77 (1996)
11. Morton, A.B., Mareels, I.M.Y.: An efficient brute-force solution to the network reconfiguration problem. IEEE Trans. on Power Delivery 15(3), 996–1000 (2000)
12. Nishizeki, T., Vygen, J., Zhou, X.: The edge-disjoint paths problem is NP-complete for series-parallel graphs. Discrete Applied Math. 115, 177–186 (2001)
13. Takamizawa, K., Nishizeki, T., Saito, N.: Linear-time computability of combinatorial problems on series-parallel graphs. J. Assoc. Comput. Mach. 29, 623–641 (1982)
14. Teng, J.-H., Lu, C.-N.: Feeder-switch relocation for customer interruption cost minimization. IEEE Trans. on Power Delivery 17(1), 254–259 (2002)

A Cutting Plane Heuristic Algorithm for the Time Dependent Chinese Postman Problem

Jinghao Sun[1,2,⋆], Yakun Meng[1,2], and Guozhen Tan[3]

[1] School of Information Science and Engineering, Northeastern University, Shenyang, China
[2] School of Computer and Communication Engineering, Northeastern University, Qinhuangdao, China
[3] School of Computer Science and Technology, Dalian University of Technology, Dalian, China
jhsun@mail.dlut.edu.cn

Abstract. In this paper we describe a cutting plane heuristic algorithm for the Time Dependent Chinese Postman Problem. This algorithm is based on the facial structure of the time dependent postman polyhedron and on the simplex method. The facial structure investigated here provides cutting planes that can be separated polynomially using a maximum flow algorithm, and the simplex method exhibits the linear programming relaxation solutions, based on which two upper bound heuristics are designed. Computational results show that the cutting planes can improve the lower bound of the formulation substantially, and that the best upper bound obtained by the two stage heuristic is always very close to the lower bound in most cases.

Keywords: Chinese Postman Problem, Time Dependent, Polyhedral Theory, Cutting Plane, Heuristic Algorithm.

1 Introduction

Let $D(V, A)$ be a connected digraph, where V is the set of vertices, A is the set of arcs and with each arc $(i, j) \in A$ is associated a time dependent travel time $D_{ij}(t_i)$ starting at time t_i. Let $v_1 \in V$ be the origin vertex and t_1 be the starting time, the Time Dependent Chinese Postman Problem (TDCPP) aims to find a tour starting at v_1 and at time t_1 and traversing each arc at least once such that the total travel time is minimized.

TDCPP problem is a variant of the classical Chinese Postman Problem (CPP), which is a well known problem in graph theory. As we know, the classical CPP problems are of great practical importance to software testing optimization [1]. Similarly, the TDCPP problem presented here is motivated from test sequence optimization based on hybrid automaton [2]. The hybrid automaton, where the delay time of transition from state s_i to s_j is a function $D_{ij}(t_i)$ of the arrival time t_i at s_i, can be easily treated as a dynamic directed network $D(V, A)$ with

⋆ Corresponding author.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 163–174, 2013.

$D_{ij}(t_i)$ as the time dependent travel time of arc $(i,j) \in A$. As each state s_i corresponds to the vertex v_i in V, and each transition from s_i to s_j corresponds to the arc (i,j) in A, the optimal test sequence checking all transitions on the hybrid system can be equivalently cast as a minimal time TDCPP-tour that traverses all the arcs in time dependent network D.

The TDCPP has been proved to be $\mathcal{NP}$-hard, even if it verifies the Eulerian and First In First Out (FIFO) property [3]. Nevertheless, small to medium-sized instances verifying the FIFO property can be solved to proven optimality by the branch and bound algorithm and dynamic programming algorithm proposed in [3,4], both of which are based on the "arc - shortest path" alternative sequence.

However, at present, the most promising solution theory appears to be polyhedral theory, based on which the valid inequalities can be used as cutting planes to strengthen a linear programming (LP) relaxation. In this paper, an integer linear programming (ILP) formulation for the TDCPP that need not verify FIFO property is proposed, and the associated polyhedral results are investigated, and a cutting plane heuristic algorithm is developed. In this algorithm, a new family of facet defining inequalities are used as cutting planes, and the maximum flow algorithm is designed as the core process of cutting plane separation strategy. Furthermore, two upper bound heuristics are also designed to terminate the cutting plane procedure whenever the current LP solution is fractional and violates no inequality. Computational results show that the lower bound obtained by adding cutting planes improves substantially the LP relaxation bound of the original formulation, and that the two stage heuristic always gets a better upper bound which is very close to the value of the lower bound.

The rest of the paper is organized as follows. Section 2 introduces the ILP formulation for TDCPP, and Some results on the TDCPP polyhedron are presented in Section 3. Section 4 describes the cutting plane heuristic algorithm involving the facet inducing inequality separation strategy and two upper bound heuristics. Computational results and concluding remarks are made in the last section.

2 Problem Formulation

In this section we formulate the TDCPP problem as an integer programming model which is motivated by the previous works of Wang and Wen [5]. They focused on a variant of Chinese postman problem with time windows, namely, time constrained Chinese postman problem (TCCPP), and pointed that the traditional CPP integer programming model which only provides a network structure but does not present how to travel such that the minimum travel time can be obtained cannot incorporate time window constraints. Thus, Wang and Wen introduced the "iteration" variables to trace such CPP-tour and formulate the time windows explicitly into the traditional model. Indeed, the "iteration" is a simple circuit in which each vertex (except the origin) cannot exist twice time, and the main idea of Wang and Wen's model is to formulate the CPP-tour as a circuit sequence such that each vertex in an iteration associates with only one starting time. From this point of view, Wang and Wen's model is renamed "circuit formulation" in this paper.

Although Wang and Wen gave such a skillful formulation for the TCCPP, they did not provide any polyhedral results to show how to improve the solution. This paper focuses on the polyhedral results, and extends the circuit formulation to the time dependent Chinese postman problem whenever the travel time is a known step function $D_{ij}(t_i)$ of the starting time t_i at vertex v_i. In this way, the total period associated with each arc $(i,j) \in A$ can be divided into several time intervals. Once the time interval during which the postman starts traversing arc (i,j) is known, the travel time of arc (i,j) is a known constant.

In order to introduce our circuit formulation, We first list the notations used in the formulation as follows: n is the number of vertices including the origin; m is the number of arcs in A; K is the maximal number of circuits in the TDCPP-tour; H is the number of time intervals associated with each arc $(i,j) \in A$; B is a large number; T_{ij}^h is the upper bound of the hth interval associated with arc $(i,j) \in A$ $(h = 1, \cdots, H)$; D_{ij}^h is the travel time of arc (i,j) starting at v_i during the hth time interval. x_{ij}^k equals to 1 if arc (i,j) is traversed in the kth circuit of the TDCPP-tour, or 0, otherwise; $\delta_{ij}^{k,h}$ equals to 1 if arc (i,j) in the kth circuit is traversed during the kth time interval, or 0, otherwise; t_i^k = starting time of some arc (i,j) traversed in the kth circuit of the TDCPP-tour. In particular, t_1^1 equals the starting time t_1, and t_1^{K+1} is the ending time of the TDCPP-tour. With the above notations the TDCPP may be formulated as follows:

Min
$$\sum_{(i,j)\in A} \sum_{k=1}^{K} \sum_{h=1}^{H} D_{ij}^h \delta_{ij}^{k,h} \tag{1}$$

Subject to

$$\sum_{(i,j)\in A} x_{ij}^k = \sum_{(j,i)\in A} x_{ji}^k \qquad \forall i = 1, \cdots, n; k = 1, \cdots, K \tag{2}$$

$$\sum_{k=1}^{K} x_{ij}^k \geq 1 \qquad \forall (i,j) \in A \tag{3}$$

$$t_j^k - t_i^k \geq D_{ij}^h \delta_{ij}^{k,h} \; \forall (i,j) \in A, j \neq 1; k = 1, \cdots, K; h = 1, \cdots, H \tag{4}$$

$$t_1^{k+1} - t_i^k \geq D_{i1}^h \delta_{i1}^{k,h} \; \forall (i,1) \in A; k = 1, \cdots, K; h = 1, \cdots, H \tag{5}$$

$$\sum_{h=1}^{H} \delta_{ij}^{k,h} = x_{ij}^k \quad \forall (i,j) \in A; k = 1, \cdots, K \tag{6}$$

$$t_i^k + B(\delta_{ij}^{k,h} - 1) \leq T_{ij}^h \; \forall (i,j) \in A; \; k = 1, \cdots, K; h = 1, \cdots, H \tag{7}$$

$$t_i^k \geq T_{ij}^{h-1} \delta_{ij}^{k,h} \quad \forall (i,j) \in A; \; k = 1, \cdots, K; h = 1, \cdots, H \tag{8}$$

$$x_{ij}^k = \{0,1\} \qquad \forall (i,j) \in A; k = 1, 2, \cdots, K; \tag{9}$$

$$\delta_{ij}^{k,h} = \{0,1\} \quad \forall (i,j) \in A; k = 1, 2, \cdots, K; h = 1, \cdots, H \tag{10}$$

$$t_i^k \geq 0 \qquad \forall i = 1, 2, \cdots, n; k = 1, 2, \cdots, K+1; \tag{11}$$

The objective function (1) minimizes the total travel time of the TDCPP-tour. Constraint (2) ensures that all the vertices must be symmetric. Constraint (3)

states that each arc must be passed at least once. Given arbitrary time interval h, circuit index k and arc (i,j), Constraints (4) and (5) can calculate the travel time of arc (i,j) when the TDCPP-tour traverses arc (i,j) in the kth circuit C_k starting at vertex v_i during time interval h. Constraint (6) ensures that if the TDCPP-tour traverses arc (i,j) in the kth circuit, then it must depart from v_i within one time interval. If the departure time t_i^k of v_i belongs to the hth time interval of (i,j), then it is necessary to check whether t_i^k is in the range from the lower bound T_{ij}^{h-1} to the upper bound T_{ij}^h, which are guaranteed in constraints (7) and (8) respectively.

3 Results on TDCPP Polyhedron

In this section, we summarize some polyhedral results for the TDCPP, based on which our cutting plane heuristic algorithm is developed.

According to the circuit formulation mentioned above, every TDCPP-tour in D can be formulated as an array of K circuits. We can associate with this circuit array an integer vector $x = (x_1, \cdots, x_k, \cdots, x_K)$, the kth of which is also an integer vector denoted as $x_k = (x_{ij}^k : (i,j) \in A) \in \mathbb{B}^A$, where x_{ij}^k indicates whether arc $(i,j) \in A$ is traversed in the kth circuit of the TDCPP-tour ($k = 1, \cdots, K$). It is evident that the incidence vector x of each circuit array satisfies Constraints (2) and (3) and, conversely, each feasible solution of system (2) and (3) is the incidence vector of a circuit array. Therefore, we use Constraints (2) and (3) which have a strong combinatorial structure to define the polytope of circuit array (CA), and denote by $\mathscr{X}$ the set of the feasible solutions for the CA polytope:

$$\mathscr{X} = \{x \in \mathbb{B}^{A \times K} : x \texttt{ satisfies (2) and (3)}\}$$

This representation yields to the definition of $P_{CA}(D)$ as the convex hull of all the feasible solutions in the CA polytope: $P_{CA}(D)$=conv($\mathscr{X}$). The polyhedral results of $P_{CA}(D)$ are listed as follows.

3.1 Affinely Independent TDCPP-Tours in CA Polytope

The set of affinely independent TDCPP-tours in $P_{CA}(D)$ can be induced from the arrays of the linearly independent circuits. Note that the maximal number of the linearly independent circuits in the strong connected graph D is $m-n+1$ [7], which can be treated as the value of of the maximal circuit number K in the circuit formulation. For convenience's sake, we use $m - n + 1$ and K interchangeably, and let the maximal linearly independent circuit group be $\mathscr{C} = \{C_1, \cdots, C_K\}$. Denote by $\mathscr{X}^*$ the set of feasible solutions corresponding to the full array of K linearly independent circuits in $\mathscr{C}$: $\mathscr{X}^* \subset \mathscr{X}$. Without loss of generality, let the circuit array be $(C_K, C_{K-1} \cdots, C_1)$, then the associated solution x in $\mathscr{X}^*$ can be written as $(x_1^K, x_2^{K-1}, \cdots, x_K^1)$, where x_k^i expresses that the kth incidence vector x_k in solution x associates with the circuit $C_i \in \mathscr{C}$. It is evident that the cardinality of $\mathscr{X}^*$ is $K!$. However, the number of affinely independent TDCPP-tours in $\mathscr{X}^*$ is much less, which will be given in the following theorem.

Theorem 1. *There are* $(K-1)^2+1$ *affinely independent TDCPP-tours in* $\mathscr{X}^*$.

Proof. When $K = 1$, there is only one circuit in D. It is easy to see that the TDCPP-tour in D is unique, which by itself is affinely independent.

When $K = l$, assume that there are $(l-1)^2+1$ affinely independent TDCPP-tours in $\mathscr{X}^*$, the set of which is denoted as $\mathscr{X}_l^*$.

When $K = l+1$, we will construct l^2+1 affinely independent TDCPP-tours $\mathscr{X}_{l+1}^*$ as follows.

Firstly, construct $(l-1)^2+1$ TDCPP-tours, the set of which is denoted as $\mathscr{X}_{l+1}^{l+1}$, by appending C_{l+1} to each TDCPP-tour $x \in \mathscr{X}_l^*$. It is evident that these $(l-1)^2+1$ TDCPP-tours are affinely independent, since $\mathscr{X}_l^*$ is affinely independent.

Secondly, for each $i = 1, \cdots, l$, denote by $x^i = (x_1^j, \cdots, x_l^i)$ a TDCPP-tour in $\mathscr{X}_l^*$ with C_i as its lth circuit. Another l affinely independent TDCPP-tours, the set of which is denoted as $\mathscr{X}_{l+1}^l$, can be obtained by transforming each TDCPP-tour $x^i = (x_1^j, \cdots, x_l^i)$ to a new TDCPP-tour with $l+1$ circuits: $(x_1^j, \cdots, x_l^{l+1}, x_{l+1}^i)$ $(i = 1, \cdots, l)$. It is easy to see that the last circuit of the ith TDCPP-tour in $\mathscr{X}_{l+1}^l$ is C_i. Therefore, the incidence matrix of $\mathscr{X}_{l+1}^l$ is non-singular since these l circuits $C_1, \cdots, C_l$ are linearly independent. Furthermore, note that the lth circuit of each TDCPP-tour $x \in \mathscr{X}_{l+1}^l$ is C_{l+1}, then the incidence matrix of $\mathscr{X}_{l+1}^{l+1}$ and $\mathscr{X}_{l+1}^l$ is also non-singular because of the linearly independence of the $l+1$ circuits $C_1, \cdots, C_{l+1}$.

Finally, we will find the last $l-1$ affinely independent TDCPP-tours in $\mathscr{X}_{l+1}^*$. To construct the ith TDCPP-tour, we transform a TDCPP-tour $x = (x_1^p, \cdots, x_i^i, \cdots, x_l^q)$ in $\mathscr{X}_l^*$ whose ith circuit is C_i to the corresponding TDCPP-tour with $l+1$ circuits $(x_1^p, \cdots, x_i^{l+1}, \cdots, x_l^q, x_{l+1}^i)$ $(i = 1, \cdots, l-1)$. Note that each ith TDCPP-tour constructed above is marked by C_{l+1} as its ith circuit, it is evident that the incidence matrix of $\mathscr{X}_{l+1}^*$ is non-singular, and, hence, the l^2+1 TDCPP-tours in $\mathscr{X}_{l+1}^*$ are affinely independent.

3.2 Dimension of CA Polytope

Based on the affinely independent TDCPP-tours found in $\mathscr{X}^*$, the dimension of $P_{CA}(D)$ is shown in the theorem below.

Theorem 2. $Dim(P_{CA}(D)) = (m-n+1)^2$, *if and only if the minimum number of circuits covering all the arcs is less than* $m-n+1$.

Proof. Sufficiency. Firstly, determine the upper bound of $Dim(P_{CA}(D))$. In the $(m-n+1)m$-dimensional solution space of $P_{CA}(D)$, each incidence vector of TDCPP-tour x satisfies $(m-n+1)n$ equations in (2). It is easy to prove that $Rank(A^E, b^E) = (m-n+1)(n-1)$, where (A^E, b^E) denotes the coefficient matrix of equation set (2). Therefore, $Dim(P_{CA}(D)) \leq (m-n+1)^2$. Secondly, determine the lower bound of $Dim(P_{CA}(D))$. According to Theorem 1, we can find a set of $(m-n)^2+1$ affinely independent TDCPP-tours in $\mathscr{X}^*$, which is denoted as $\mathscr{X}_K^*$. As the minimum number of circuits covering all the arcs is

less than $m-n+1$, there must exist a TDCPP-tour $x \in \mathscr{X}$ which contains $m-n$ circuits in D. Without loss of generality, let this TDCPP-tour be $x = (x_1^1, x_2^2, \cdots, x_{m-n}^{m-n})$, where x_k^k expresses that the kth circuit of TDCPP-tour x is $C_k \in \mathscr{C}$. Then, other $m-n+1$ affinely independent TDCPP-tours can be found in $\mathscr{X}$: $\mathscr{X}_\emptyset = \{(x_1^1, \cdots, x_{m-n}^{m-n}, \emptyset), (x_1^1, \cdots, \emptyset, x_{m-n+1}^{m-n}), \cdots, (\emptyset, x_1^1, \cdots, x_{m-n+1}^{m-n})\}$, where $\emptyset$ denotes the void circuit with no arc. Similarly, the last $m-n$ affinely independent TDCPP-tours can be found in $\mathscr{X}$: $\mathscr{X}_C = \{(x_1^1, \cdots, x_{m-n}^{m-n}, x_{m-n+1}^1), (x_1^1, \cdots, x_{m-n}^1, x_{m-n+1}^{m-n}), \cdots, (x_1^1, x_2^1, \cdots, x_{m-n+1}^{m-n})\}$. Note that the kth TDCPP-tour in $\mathscr{X}_\emptyset$ is marked by $\emptyset$ as its kth circuit ($k = 1, \cdots, m-n+1$), and the kth TDCPP-tour in $\mathscr{X}_C$ contains a duplicate C_1 as its kth circuit. It is evident that the incidence matrix of $\mathscr{X}_K^*$, $\mathscr{X}_\emptyset$ and $\mathscr{X}_C$ is non singular. Hence there are $(m-n+1)^2+1$ affinely independent TDCPP-tours in $\mathscr{X}$.

Necessity. If the minimum number of circuits covering all the arcs of D is equal to $m-n+1$, then $\mathscr{X}^* = \mathscr{X}$. According to Theorem 1, we can find only $(m-n)^2+1$ affinely independent TDCPP-tours in $\mathscr{X}$, the number of which does not reach $(m-n+1)^2$. Thus, if $Dim(P_{CA}(D)) = (m-n+1)^2$, then the minimum number of circuits covering all the arcs of D is less than $m-n+1$.

3.3 Facet Defining Inequalities for the CA Polytope

A feasible solution $x \in \mathscr{X}$ for the CA polytope should satisfy the following inequality if it corresponds to the optimal TDCPP-tour of TDCPP.
k-cut inequality:

$$\sum_{(i,j)\in\omega_k} x_{ij}^k \geq 1 \qquad k = 1, 2, ..., K; \tag{12}$$

where ω_k expresses the cut set of contracted graph D^k obtained by contracting the arcs of circuits except C_k in $\mathscr{C}$ into a single vertex (The arc contracting procedure is introduced in [7] for more details). Contracted graph D^k is divided into two subgraphs by ω_k, denoted as D_1^k and D_2^k, where D_1^k contains the origin vertex of C_k and has no strongly connected component.

Constraint (12) ensures that each non-empty circuit C_k in the TDCPP-tour must have at least one arc which will not be traversed in other circuits, that is to say, the arc set of C_k, denoted A_k, should satisfy $A_k - \cup_{l\neq k} A_l \neq \emptyset$. In other words, if there exists a circuit in the TDCPP-tour whose arcs are all traversed in other circuits then we can get rid of this redundant circuit and obtain a better TDCPP-tour with smaller total travel time. Thus, Constraints (12) can help us to exclude a lot of non-optimal TDCPP-tours with redundant circuits.

Theorem 3. *Constraint (12) induces a facet defining inequality of CA polytope if and only if ω_k is the minimal cut set in contracted graph D^k for arbitrary $k = 1, 2, ..., K$.*

Proof. Sufficiency. According to Theorem 1, $(m-n)^2+1$ affinely independent TDCPP-tours can be induced from the arrays of K linearly independent circuits in $\mathscr{C}$, the set of which is denoted as $\mathscr{X}_K^*$. Obviously, the kth circuit C_k of each

TDCPP-tour in $\mathscr{X}_K^*$ must satisfy $A_k - \cup_{l\neq k} A_l \neq \emptyset$, otherwise C_k could be represented by other circuits in $\mathscr{C}$, which is contrary to the definition of linearly independent circuit set $\mathscr{C}$. Therefore, C_k satisfies $|A_k \cap \omega_k| = 1$, which implies that each tour in $\mathscr{X}_K^*$ satisfies (12) as an equality. Moreover, as the minimal number of circuits covering all the arcs is less than $m-n+1$, there must exist a TDCPP-tour $x \in \mathscr{X}$ which contains $m-n$ circuits in D. Without loss of generality, let this TDCPP-tour be $x = (x_1^1, x_2^2, \cdots, x_{m-n}^{m-n})$. Then, for an arbitrary $k = 1, \cdots, K$, another $2(m-n)$ affinely independent TDCPP-tours can be found: $\mathscr{X}'_\emptyset = \{(x_1^1, \cdots, x_{m-n}^{m-n}, \emptyset), (x_1^1, \cdots, x_k^k, \emptyset, x_{m-n+1}^{m-n}), (x_1^1, \cdots, \emptyset, x_k^k, x_{m-n+1}^{m-n}) \cdots, (\emptyset, x_1^1, \cdots, x_{m-n+1}^{m-n})\}$ and $\mathscr{X}'_C = \{(x_1^1, \cdots, x_{m-n}^{m-n}, x_{m-n+1}^1), (x_1^1, \cdots, x_{m-n}^1, x_{m-n+1}^{m-n}), \cdots, (x_1^1, x_2^1, \cdots, x_{m-n+1}^{m-n})\}$. It is easy to prove that the kth circuit of each TDCPP-tour x in $\mathscr{X}'_\emptyset$ and $\mathscr{X}'_C$ must satisfy $|A_k \cap \omega_k| = 1$ since ω_k is the minimal cut set in D^k. Thus, the TDCPP-tour x satisfies (12) as an equality by direct substitution for an arbitrary k. Furthermore, the incidence matrix of $\mathscr{X}_K^*$, $\mathscr{X}'_\emptyset$ and $\mathscr{X}'_C$ is non-singular, and, hence, the $(m-n+1)^2$ TDCPP-tours found above are affinely independent.

Necessity. Assume that ω_k is not minimal, that is to say, $|A_k \cap \omega_k| > 1$ for some circuit C_k, which implies the Constraint (12) could not be satisfied as an equality for some TDCPP-tour whose kth circuit is C_k. Therefore, there are no more than $(m-n+1)^2-1$ affinely independent tours in $\mathscr{X}$ satisfying Constraint (12) as an equality.

To check the effect of the facet defining inequalities (12), we have designed an LP-based cutting plane algorithm that adds inequalities (12) as cutting planes. Our computational experiments in Section 5 show that the inequalities (12) can provide a better lower bound.Moreover, as our next theorem shows, inequalities (12) possesses another nice property.

Theorem 4. *The separation problem for the inequalities (12) can be solved in polynomial time.*

Proof. Given a (rational) vector $x^* \in \mathbb{Q}^{mK}$, the separation problem for inequalities (12) is to determine whether x^* satisfies inequalities (12). For each arc $(i,j) \in A$ and $k = 1, \cdots, K$, define the weight

$$w_{ij} = \begin{cases} B & : \ \sum_{r\neq k} x_{ij}^r \geq 1 \\ x_{ij}^k & : \ else \end{cases}$$

where B is a big number. Obviously, the arc will be associated with a weight B if it has been traversed in any circuits except C_k of the TDCPP-tour, thus, the contracted graph $D^k(V^k, A^k)$ mentioned in Section 3.3 can be obtained by contracting each arc with weight B into a single vertex. In order to determine the minimum cut ω_k, we should construct a new graph $\bar{D}^k(\bar{V}^k, \bar{A}^k)$ at first by modifying D^k as follows. 1) introduce an extra vertex ν to D^k, and let $\bar{V}^k = V^k \cup \{\nu\}$. 2) Denote by v_s the origin vertex of C_k, and for every arc (i,s) of D^k which points to the origin v_s, attach vertex v_i to the extra vertex ν, and delete arc (i,s), that is to say, $\bar{A}^k = A^k \cup \{(i,\nu)|(i,s) \in A^k\} - \{(i,s)|(i,s) \in A^k\}$.

It is evident that each cut set $\omega_{s\nu}$ that separates origin v_s and extra vertex ν in $\bar{D}^k$ can be equivalently cast as a corresponding cut ω_k in D^k, denoted as $\omega_{s\nu} \sim \omega_k$. Since for each $k = 1, \cdots, K$ and any cut set $\omega_{s\nu}$, x^* satisfies inequality (12): $\sum_{(i,j)\in\omega_k} x_{ij}^k \geq 1 \Leftrightarrow \sum_{(i,j)\in\omega_{s\nu}} w_{ij} \geq 1$ if $\omega_{s\nu} \sim \omega_k$, one can see that the separation problem for the k-cut inequalities (12) reduces to the problem of determining a minimum cut set of $\bar{D}^k$ with respect to the nonnegative weight function w. Padberg and Rao (1982) have shown that the latter is polynomially solvable. Hence the theorem is proved.

3.4 Further Strong Valid Inequalities for TDCPP

The timing constraint (7) involves a "big" number B and generally results in weak LP relaxations. Thus two strong valid inequalities are also proposed here, the form of which is listed as follows.

$$\delta_{ij}^{k,h} + \sum_{(j,l)\in A} \sum_{p\in A_{ijl}^h} \delta_{jl}^{k,p} \leq 1 \; \forall (i,j) \in A; k = 1, \cdots, K; h = 1, \cdots, H \tag{13}$$

$$\sum_{(i,j)\in A} \sum_{h\in B_{ijl}^p} \delta_{ij}^{k,h} + \delta_{jl}^{k,p} \leq 1 \; \forall (j,l) \in A; k = 1, \cdots, K; h = 1, \cdots, H \tag{14}$$

where

$$A_{ijl}^h = \{p \,|T_{jl}^p < T_{ij}^{h-1} + D_{ij}^h \quad or \quad T_{jl}^{p-1} > T_{ij}^h + D_{ij}^h + \texttt{DIFF}\}$$

$$B_{ijl}^p = \{h \,|T_{jl}^p < T_{ij}^{h-1} + D_{ij}^h \quad or \quad T_{jl}^{p-1} > T_{ij}^h + D_{ij}^h + \texttt{DIFF}\}$$

and

$$DIFF = max\{0, D_{jl}^{p-1} - D_{jl}^p\}$$

Constraint (13) ensures that if the TDCPP-tour traverses arc (i, j) starting at v_i during time interval h, then for any successor arc (j, l) of (i, j), it is impossible to be traversed during the time intervals in A_{ijl}^h which involves a term DIFF introduced by Malandraki [6]. The inclusion of the term DIFF allows the travel time step function on (v_j, v_l) to behave as if it was a piecewise linear continuous function when the travel time in period p is less than that in the preceding period $p - 1$, as discussed in [6]. With this technique, Constraint (14) which is the following analog of Constraint (13) restricts the time intervals of all the predecessors of each arc in the TDCPP-tour.

4 Description of the Cutting Plane Heuristic Algorithm

We now describe our algorithm below to solve TDCPP problem.

Step 1 (Initialization). Set $\mathscr{P} := \{(x, \delta, t) = ((x_{ij}^k : (i,j) \in A; k = 1, \cdots, K), (\delta_{ij}^{k,h} : (i,j) \in A;\ k = 1, \cdots, K; h = 1, \cdots, H), (t_i^k : v_i \in V;\ k = 1, \cdots, K+1)) \,|(x, \delta, t)$ `satisfies (2-8), (13) and (14)`$\}$.

This step initializes the polyhedron $\mathscr{P}$ with the LP relaxation of the circuit formulation. The number of constraints of the linear system describing $\mathscr{P}$ (not counting the nonnegativity constraints) is $(5mH+m+n)K+m$. LPs of this type can be solved in polynomial time theoretically (by using the ellipsoid method).

Step 2 (Solving the LP). Solve the linear program

$$\text{Min} \quad \sum_{(i,j)\in A} \sum_{k=1}^{K} \sum_{h=1}^{H} D_{ij}^{h} \delta_{ij}^{k,h}$$

$$\text{Subject to} \quad (x, \delta, t) \in \mathscr{P}$$

Step 3 (Cutting plane separation). Find the k-cut inequalities (12) which are violated by the optimal LP solution, say (x^*, δ^*, t^*), of step 2.

Such inequalities, if they exist, induce cutting planes which chop off (x^*, δ^*, t^*). One can check in polynomial time whether the current LP optimal solution violates the k-cut inequalities (12) according to Theorem 4.

Step 4 (Adding cutting plane). If some inequalities in (12) violated by (x^*, δ^*, t^*) are found, then add them to the linear system. Let $\mathscr{P}$ be the polyhedron determined by the resultant linear system and go to Step 2.

Step 5 (Constructing an approximate tour). When the current LP solution is fractional and violates no inequality. Apply a heuristic that uses the fractional LP solution to construct an approximate TDCPP-tour, calculate an upper bound and stop. There are two upper bound heuristics used in this step: One Stage Heuristic and Two Stage Heuristic.

One Stage Heuristic
Input: sub-vector x^* of the current optimal LP solution.
Output: an approximate tour $P = (C_1, C_2, \cdots, C_K)$.

1. Denote by A_k the associated arc set of the kth circuit C_k in TDCPP-tour P, then set $A_k = \emptyset$ $(k = 1, \cdots, K)$.
2. For each arc $(i, j) \in A$ repeat the following step (a-c).
   (a) For $(i, j) \in A$, randomly select a k by probability $x_{ij}^k / \sum_{k=1}^{K} x_{ij}^k$.
   (b) Determine whether arcs in $A_k \cup \{(i, j)\}$ can be contained in a single circuit, if not, go to step (a).
   (c) Insert arc (i, j) into arc set A_k.
3. When each arc in A is assigned to some circuit of the TDCPP-tour P, construct each kth circuit C_k of P by connecting the arcs in A_k $(k = 1, \cdots, K)$.
4. Traverse the TDCPP-tour P starting from origin v_1 at time t_1, and calculate the total travel time of P.

Two stage Heuristic
Input: sub-vector (x^*, δ^*) of the current optimal LP solution.
Output: an approximate tour P.
Stage One: construct an unicursal graph D_E.

1. Let $D(V, A, w)$ be the weighted strongly connected network, and for each arc $(i, j) \in A$, the associated weight $w_{ij} \in w$ is calculated as $\sum_{k=1}^{K} x_{ij}^k$.

2. Let I be the set of vertices v_i for which the number of incoming arcs exceeds the number of outgoing arcs by s_i and J, the set of vertices v_j for which the number of outgoing arcs exceeds the number of incoming arcs by d_j. Construct a capacity-cost network $D'(V', A', c', w')$ with source v_s and sink v_t: $V' = V \cup \{v_s, v_t\}$, $A' = A \cup \{(s,i)|v_i \in I\} \cup \{(j,t)|v_j \in J\}$; for all $(i,j) \in A$, let $c'_{ij} = +\infty$, $w'_{ij} = w_{ij}$; for all $v_i \in I$, let $c'_{si} = s_i$, $w'_{si} = 0$; for all $v_j \in J$, let $c'_{jt} = -d_j$, $w'_{jt} = 0$;
3. Use the minimum mean cycle canceling algorithm [8] to find a minimum cost flow $f = f_{ij}|(i,j) \in A'$ from v_s to v_t in D'.
4. Replicate each arc $(i,j) \in A$ f_{ij} times by introducing copies with the same weight w_{ij}, and the unicursal graph D_E is obtained.

Stage Two: determine an Eulerian cycle (the TDCPP-tour P) on D_E.

1. Set $v_i = v_1$, $t_i = t_1$.
2. Denote by $N(i)$ the neighbors of vertex v_i, and randomly select an arc $(i,j) \in A$ as the next arc traversed in the TDCPP-tour P by probability

$$p_{ij} = \frac{\sum_{k=1}^{K} \delta_{ij}^{k,h}}{\sum_{v_j \in N(i)} \sum_{k=1}^{K} \delta_{ij}^{k,h}} \qquad \forall v_j \in N(i)$$

   where $T_{ij}^{h-1} \leq t_i < T_{ij}^{h}$ is satisfied for each $\delta_{ij}^{h,k}$.
3. Traverse arc (i,j) starting from vertex v_i at time t_i, and erase arc (i,j).
4. Set $v_i = v_j$, $t_i = t_j$, and repeat step ii starting from the other extremity of the deleted arc or stop and save the total travel time t_i of P if all arcs have been deleted.

5 Computational Results

This section summarizes the computational results obtained by the cutting plane heuristic algorithm described above. This algorithm was coded in C++ using the LINGO 8.0 library. The code ran on a PC with a Pentium processor at 2.2GHz with 1G RAM. Several randomly generated instances were used for our computational study, and the generation procedure was described in [3].

The computational results obtained by the cutting plane heuristic algorithm are summarized in table 1. The column headings are defined as follows: *Inst.*: instance identifier; $|V|$: number of vertices of the original graph; $|A|$: number of arcs of the original graph; H: number of time intervals associated with each arc; *Cuts*: total number of k-cut inequalities; LP: the LP relaxation bound of the original formulation including (13) and (14); LB: the lower bound obtained by the cutting plane algorithm using the facet defining inequalities (12); $UB1$: the upper bound obtained by the one stage heuristic; $UB2$: the upper bound obtained by the two stage heuristic; $G1$: the relative gap between the best upper bound $UB = min(UB1, UB2)$ and LP (computed as $((UB - LP)/LP) \times 100$); $G2$: the relative gap between the best upper bound UB and the lower bound LB (computed as $((UB - LB)/LB) \times 100$).

Table 1. Computational result of the test instances

$Inst.$	$\|V\|$	$\|A\|$	H	$[-R, R]$	LP	LB	$UB1$	$UB2$	$G1(\%)$	$G2(\%)$
A1-1	10	30	2	[-10,10]	991	1019	1147	1053	6.26	3.34
A1-2	15	45	2	[-10,10]	1484	1523	1645	1629	9.77	6.96
A1-3	20	40	2	[-10,10]	1330	1411	1530	1498	12.63	6.17
A1-4	25	50	2	[-10,10]	1553	1583	1889	1739	11.98	9.85
B1-1	10	30	3	[-10,10]	1027	1114	1176	1124	9.44	0.90
B1-2	15	45	3	[-10,10]	1530	1699	1901	1793	17.19	5.53
B1-3	20	40	3	[-10,10]	1565	1754	2055	1812	15.78	3.31
B1-4	25	50	3	[-10,10]	1434	1680	1894	1779	24.06	5.89
C1-1	10	30	4	[-10,10]	1052	1171	1337	1286	22.24	9.82
C1-2	15	45	4	[-10,10]	1575	1809	2113	1925	22.22	6.41
C1-3	20	40	4	[-10,10]	1519	1704	2028	1832	20.61	7.51
C1-4	25	50	4	[-10,10]	1527	1805	1980	1850	21.15	2.49
A2-1	10	30	2	[-20,20]	1923	1952	2702	2490	29.49	27.56
A2-2	15	45	2	[-20,20]	2889	2914	3608	3608	24.89	23.82
A2-3	20	40	2	[-20,20]	2577	2735	3299	3288	27.59	20.22
A2-4	25	50	2	[-20,20]	3253	3327	4113	4060	24.81	22.03
B2-1	10	30	3	[-20,20]	1741	1831	2580	2375	36.42	29.71
B2-2	15	45	3	[-20,20]	2618	2730	3577	3512	34.15	28.64
B2-3	20	40	3	[-20,20]	2143	2264	2938	2752	28.42	21.55
B2-4	25	50	3	[-20,20]	3100	3124	3880	3880	25.16	24.20
C2-1	10	30	4	[-20,20]	1869	1948	2755	2494	33.44	28.03
C2-2	15	45	4	[-20,20]	2791	3017	3940	3645	30.60	20.82
C2-3	20	40	4	[-20,20]	2433	2586	3488	3407	40.03	31.75
C2-4	25	50	4	[-20,20]	3092	3424	4101	4003	29.46	16.91

Results presented in table 1 indicate that the best upper bound UB obtained by the cutting plane heuristic algorithm is always very close to the value of lower bound LB. In our test problems, the average gap $G2$ is almost less than 15.14%. This performance can be explained to a large extent by the facet defining inequalities (12).

Meanwhile, we also find that changes of fluctuation interval might affect the gap $G2$. For these instances with fluctuation interval [-10,10], the relative gap $G2$ attains 5.68% while it rises to 24.6% for those instances with fluctuation interval [-20,20]. In addition, nearness of the approximate tours to lower bounds also shows the efficiency of the two upper bound heuristics of Section 4. We observe that the best upper bounds of all our instances are obtained by the two stage upper bound heuristic. This empirically shows that the two stage heuristic based on both x^* and δ^* is more effective than the one stage heuristic which is only based on sub-vector x^* of the LP relaxation solution, that is to say, the LP relaxation solution plays an important role in the upper bound heuristic.

The lower bound LB obtained by adding cutting planes improves substantially the LP relaxation bound LP of the original formulation for all instances. By comparing columns $G1$ and $G2$, one can appreciate how much our cutting planes have contributed to the relative gaps dramatically. As shown in Table 1, the percentage difference G between gap $G2$ and $G1$ is 8.1% on average. Moreover, G rises with increasing time interval and fluctuation interval. As shown in Table 1, G always exceeds 12% averagely for those instances with four time intervals, while it is less than 3.43% for those instances with two time intervals. Moreover, when the fluctuation interval rises from [-10,10] to [-20,20], the difference G increases from 10.43% to 5.77% on average. The results indicate that the k-cut inequalities (12) will play more important roles with more time intervals and less fluctuations.

6 Conclusion

We proposed a ILP formulation for the TDCPP problem, namely, circuit formulation, and the associated partial polyhedral structure is investigated. Moreover, we developed a cutting plane heuristic algorithm for solving the TDCPP problem based on the partial linear description of the polyhedron. The partial linear description (together with cutting plane separation strategies) provides cutting planes and hence generates better lower bound than the LP relaxation bound of the original formulation. Computational results show that the lower bound obtained by adding cutting planes improves the LP relaxation bound of the original formulation for all instances. Two heuristic algorithms based on the relaxation solutions, namely, one stage heuristic and two stage heuristic are designed to obtain the upper bound. Computational results show that the two stage heuristic always gets a better upper bound.

Acknowledgments. This research was performed at Northeastern University and was supported by the Natural Science Foundation for Young Scholars of Hebei Province of China(NO.F2013501048) and the Science and Technology Research and Development Program of Qinhuangdao(No.2012021A101). The support is gratefully acknowledged.

References

1. Chi, C., Hao, R.: Test Generation for Interaction Detection in Feature-Rich Communication Systems. Computer Networks 51, 426–438 (2007)
2. Springintveld, P.R., Vaandrager, F.: Testing timed automata. Theoretical Computer Science 254, 225–257 (2001)
3. Tan, G.Z., Sun, J.H., Meng, Y.K.: Solving the Time Dependent Chinese Postman Problem by Branch & Bound Algorithm. Information 15, 5263–5270 (2012)
4. Sun, J.H., Tan, G.Z., Qu, H.L.: Dynamic programming algorithm for the time dependent Chinese Postman Problem. Journal of Information and Computational Science 8, 833–841 (2011)
5. Wang, H.F., Wen, X.P.: Time-Constrained Chinese Postman Problems. Computers and Mathematics with Applications 44, 375–387 (2002)
6. Malandraki, C., Daskin, M.S.: Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms. Transportation Science 26, 185–200 (1992)
7. Bang, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications. Springer, London (2001)
8. Goldberg, A.V., Tarjan, R.E.: Finding minimum-cost circulations by canceling negative cycles. Journal of the ACM 36, 873–886 (1989)

Zero-Visibility Cops and Robber Game on a Graph

Dariusz Dereniowski[1,⋆], Danny Dyer[2], Ryan M. Tifenbach[2,⋆⋆], and Boting Yang[3]

[1] Department of Algorithms and System Modeling, Gdańsk University of Technology, Gdańsk, Poland
[2] Department of Mathematics and Statistics, Memorial University of Newfoundland, St. John's, Canada
[3] Department of Computer Science, University of Regina, Regina, Canada

Abstract. We examine the zero-visibility cops and robber graph searching model, which differs from the classical cops & robber game in one way: the robber is invisible. We show that this model is not monotonic. We also provide bounds on both the zero-visibility copnumber and monotonic zero-visibility copnumber in terms of the pathwidth.

1 Introduction

Using mobile agents to find and capture a mobile intruder is a well-studied graph theory problem. Depending on the restrictions placed on the agents and the intruder, the resulting pursuit can vary wildly. One common restriction placed on both the agents and the intruder is a speed limit; in some versions of this game, while the agents may only move along edges one at a time, the intruder may move from any position on the graph to any other along a connected path that does not contain any agents. In other versions, the agents may "jump" from a vertex to any other vertex. In still other games, one or both of the agents and the intruder have limited information about the other party's position; that is, one party or the other may only see the opposition if they are near one another, or alternatively, may never see each other until they stumble upon each other at the same vertex.

The cop and robber model was introduced independently by Winkler and Nowakowski [14] and Quilliot [15]. In this model, a slow, visible intruder (the robber) moves from vertex to adjacent vertex in a graph, while pursued by one slow, visible agent (the cop), who also moves from vertex to adjacent vertex. In these first papers, *copwin* graphs were characterised; that is, graphs where exactly one cop was sufficient to capture. Many questions have grown out of

⋆ D. Dereniowski has been partially supported by Narodowe Centrum Nauki under contract DEC-2011/02/A/ST6/00201.
⋆⋆ Corresponding author. R.M. Tifenbach has been supported by a postdoctoral fellowship from the Atlantic Association for Research in the Mathematical Sciences.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 175–186, 2013.

these papers. Recently, a characterisation of the *k-copwin* graphs has been discovered [3].

A variation of a less-studied version of this problem dates back to Tošić in 1985 [18]. This corresponds to the cop and robber model with one exception: the robber is invisible. This *zero-visibility* cops and robber model may also be taken as a particular instance of a k-visibility cops and robber problem, where both cops and robber move as in the standard Winkler-Nowakowski-Quilliot model, but the robber only becomes visible to the cops when he is at distance at most k from some cop. In these models, the analog of copnumber can be defined in different ways. Following [17] and [18], we define the zero-visibility copnumber of a graph G to be the minimum number of cops needed to guarantee capture of an invisible robber in a finite time. (Other authors [11] do not necessarily include the restriction to finite time, which works well with their application of the probabilistic method.)

The zero-visibility copnumber for paths, cycles, complete graphs and complete bipartite graphs were characterised in [18], as were graphs that are zero-visibility copwin. There are several constructions in [10] for graphs which require at most 2 cops to perform a zero-visibility search, but a characterisation remains open. An algorithm for determining the zero-visibility copnumber of a tree was given in [17,5], but the problem for general graphs is NP-complete [5]. Most recent work on these topics has been on limited (but not zero) visibility [8,9], and on the expected capture time of the robber in the zero or limited visibility case [1,9,11].

One topic that has been a mainstay of edge searching problems is *monotonicity*. Basically, a search is monotonic if, once a region has been guaranteed to be free of the robber, the cops may not move in such a way to allow the robber to re-enter that region. It is well known that edge searching is monotonic [2,13], but that connected edge searching is not [20,21]. In the original cops and robber game, it was hard to motivate a definition of monotonicity, as the robber was visible. In the zero-visibility version this becomes a natural question again. We will show that the zero-visibility copnumber is different from its monotonic equivalent, and we will discuss bounds on both of these numbers based on the pathwidth of the graph.

2 Zero-Visibility Cops and Robber

We consider a pursuit game on a graph we refer to as *zero-visibility cops & robber*. The game is played on a simple connected graph G between two opponents, referred to as the *cop* and the *robber*. The cop controls the movements of a fixed number of cop pieces and the robber player controls the movement of a single robber piece (we refer to both the players and their pieces as cops and robber). The cop player begins by placing the cops on some collection of vertices of G (more than one cop may occupy a vertex) and his opponent then places the robber on a vertex, unknown to the cop. The players then alternate turns, beginning with the cop; on each player's turn, he may move one or more of his pieces from its current vertex to an adjacent vertex (either player may leave

any or even all of his pieces where they are). The game ends with a victory for the cop player if, at any point, the robber piece and a cop piece occupy the same vertex. The robber wins if this situation never occurs. It is important to emphasise that, until he has won, the cop player has no information regarding the robber's position or moves – he cannot see the robber piece until it and a cop occupy the same vertex. On the other hand, the cop may, due to his past moves, gain some knowledge on the possible locations of the robber.

All graphs are assumed to be *simple*; any two vertices are joined by at most one edge and there are no loops (edges from a vertex to itself). We introduce the following terminology regarding this game.

For a graph G, V_G and E_G are the vertex and edge sets of G. We use the symbol $x \sim y$ to represent the fact that x and y are distinct vertices joined by the edge $xy \in E_G$ and the symbol $x \simeq y$ to represent that $x \sim y$ or $x = y$. For each $X \subseteq V_G$, the set $N[X] = \{x \in V : \exists y \in X \text{ such that } x \simeq y\}$ is the *closed neighbourhood* of X. If $X = \{x\}$ is a singleton, we use $N[x]$ rather than $N[\{x\}]$ to represent the closed neighbourhood of x. For $X \subseteq V_G$, the *boundary* of X is the set of vertices adjacent to members of X but not contained in X: $\delta(X) = \{y \notin X : \exists x \in X \text{ such that } x \sim y\} = N[X] \setminus X$.

We will make extensive use of the concept of a *walk* in a graph; however, we give walks additional structure normally not present in their definition. We define a walk in a graph to be a (possibly infinite) sequence of vertices $\alpha = (\alpha(0), \alpha(1), \ldots)$ such that for all $t \geq 0$, $\alpha(t+1) \simeq \alpha(t)$. We use walks to describe cops' and robber's movements; if a walk α corresponds to the positions of a single piece within the game – the vertex $\alpha(0)$ is the starting position of the piece and the vertex $\alpha(t)$ is the location of the piece after its controller has taken t turns.

A *strategy* on G for k cops is a finite set of walks $\mathcal{L} = \{l_i\}_{i=1}^{k}$, all of the same length T (possibly $T = \infty$). A strategy $\mathcal{L}$ corresponds to a potential sequence of turns by the cop player; each walk $l_i \in \mathcal{L}$ corresponds to the moves of one of the cop pieces. The order of a strategy is the number of cop pieces required to execute it. If a strategy has length $T < \infty$, we might imagine that the cop player forfeits if he hasn't won after T moves. We say that a strategy is *successful* if it guarantees a win by the cop player – a successful strategy results in a win for the cops regardless of the moves made by the robber. Evidently, a strategy $\mathcal{L} = \{l_i\}_{i=1}^{k}$ of length T is successful if and only if for every walk α of length T in G, there are $l_i \in \mathcal{L}$ and $t < T$ such that $\alpha(t) = l_i(t)$ or $\alpha(t) = l_i(t+1)$. (The robber must be caught at some point, either by moving onto a cop or having a cop move onto it.)

The *zero-visibility copnumber* of a connected graph G is the minimum order $c_0 = c_0(G)$ among successful strategies on G; it is the smallest number of cops required to guarantee capture of the robber.

Typically, in pursuit games of this sort, only finite strategies are considered successful – the robber must be caught in a bounded number of turns. However, the following theorem shows that if we allow infinite strategies in the zero-visibility cops & robber game, any successful strategy (as defined above) will succeed in a bounded amount of time, whether or not the strategy itself is finite.

Theorem 1. *Let G be a graph. Any infinite successful strategy on G may be truncated to obtain a finite successful strategy.*

In light of Theorem 1, we will only consider finite strategies. Moreover, we can recast this game as a *node-search* style problem. Rather than imagine an opponent, we simply keep track of all possible vertices on which the robber piece might be found, via the following construction: (1) Initially, every vertex is marked as *dirty*; (2) a dirty vertex is *cleaned* if a cop piece occupies it; and (3) in between each of the cop's turns, every clean vertex that is unoccupied and adjacent to a dirty vertex becomes dirty.

The dirty vertices are the set of all possible locations of the robber. We refer to the step in between the cop's turns where unoccupied vertices may become dirty as *recontamination*.

Let G be a graph and let $\mathcal{L}$ be a strategy of length T. For each nonnegative integer $t \leq T$, let $\mathcal{L}_t$ be the set of vertices occupied by cops after t turns by the cop player; let $\mathcal{R}_t$ be the set of vertices that are dirty immediately *before* the cop's t-th turn; and let $\mathcal{S}_t$ be the set of vertices that are dirty immediately *after* the cop's t-th turn.

In other words, at the beginning of a t-th turn, $t \geq 1$, the cops occupy the vertices in $\mathcal{L}_{t-1}$ and $\mathcal{R}_t$ are the dirty vertices (possible locations of the robber). Then, the cops move and $\mathcal{L}_t$ becomes the vertex set they occupy, and $\mathcal{S}_t$ becomes the set of vertices that are dirty. After the following robber's move $\mathcal{R}_t$ is the set of dirty vertices.

We define, somewhat arbitrarily, $\mathcal{R}_0 = V$. For $t \geq 0$, the relevant rules of the game imply that $\mathcal{S}_t = \mathcal{R}_t \setminus \mathcal{L}_t$, $\mathcal{R}_{t+1} = N[\mathcal{S}_t] \setminus \mathcal{L}_t$, and $\mathcal{L}_{t+1} \subseteq N[\mathcal{L}_t]$.

A strategy of finite length T is successful if and only if $\mathcal{S}_T$ is empty.

In a pursuit game of this sort, a topic of general interest is that of the monotonicity of strategies. Typically, a strategy is *monotonic* if recontamination never occurs. In this case, such a strategy would have

$$\mathcal{R}_0 \supseteq \mathcal{S}_0 \supseteq \mathcal{R}_1 \supseteq \mathcal{S}_1 \supseteq \ldots \supseteq \mathcal{R}_T \supseteq \mathcal{S}_T,$$

where T is the length of the strategy.

However, consider the following possible activity of a single cop piece. Let xy be an edge and suppose we are attempting to construct a strategy that cleans the graph G. If a single cop moves back and forth between x and y (that is, moves from one to the other every turn), the two vertices x and y are guarded from the robber – if the robber moves onto either while this is occurring he will be caught either immediately or on the next turn.

Considering this activity under the node-search model, although the vertices x and y are possibly being recontaminated over and over, the contamination can never "spread" through them, as they are cleaned before they can possibly recontaminate any further vertices.

We will refer to the above activity as *vibrating* on the edge xy. Further, if E is a set of edges, we say that a set of cops is vibrating on E if each is vibrating on a member of E and every member of E is thus protected. We will also occasionally refer to a set of cops vibrating on a set of vertices X; this simply means that X

is covered by some set of edges E and the cops are vibrating on E. Typically, we want a set of cops to vibrate on a matching (a set of edges that do not share any endpoints), in order to most efficiently utilise this tool.

We wish to take advantage of the above strategic element while still exploring the topic of monotonicity. Thus, we define a strategy of length T to be *weakly monotonic* if for all $t \leq T-1$, we have $\mathcal{S}_{t+1} \subseteq \mathcal{S}_t$. In a weakly monotonic strategy, every time a clean vertex is recontaminated, it is cleaned on the very next move by the cop.

The *monotonic zero-visibility copnumber* of a connected graph G is the minimum order $mc_0 = mc_0(G)$ among successful weakly monotonic strategies on G; it is the smallest number of cops required to capture the robber utilising a weakly monotonic strategy. We are exclusively interested in weakly monotonic strategies as opposed to the stronger variant, and so we will simply use the term monotonic, with the understanding that this means weakly monotonic as defined above. Clearly, we have $c_0(G) \leq mc_0(G)$ for all graphs G.

A *matching* in a graph is a set of edges such that no two are incident (share an endpoint). The *matching number*, $\nu(G)$, is the maximum size of a matching in the graph G. It is well-known that a maximum matching, and thus the matching number, can be found in polynomial time [6].

Theorem 2. *Let G be a connected graph; then, $mc_0(G) \leq \nu(G)+1$, with equality if and only if G is a complete graph on an odd number of vertices.*

A *clique* in a graph is a set of vertices that are all adjacent to each other. The *clique number* of the graph G, denoted as $\omega(G)$, is the maximum of size of a clique in G. A *complete graph* with n vertices, denoted as K_n, is a graph whose clique number is n. Theorem 3 appears in [17,18].

Theorem 3. *If G is a connected graph, then $c_0(G) \geq \frac{1}{2}\omega(G)$. Moreover, $c_0(K_n) = mc_0(K_n) = \lceil \frac{n}{2} \rceil$.*

3 Pathwidth and the Zero-Visibility Copnumber

Let G be a graph with vertex set V_G. A *path decomposition* of G is a finite sequence $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n)$ of sets $\mathcal{B}_i \subseteq V_G$ such that

1. $\bigcup_{i=1}^{n} \mathcal{B}_i = V_G$;
2. if $x \sim y$, then there is $i \in \{1, \ldots, n\}$ such that $\{x, y\} \subseteq \mathcal{B}_i$; and
3. if $1 \leq i < j < k \leq n$, then $\mathcal{B}_i \cap \mathcal{B}_k \subseteq \mathcal{B}_j$.

We refer to the sets $\mathcal{B}_i$ as *bags*. An alternate, but equivalent, formulation of the third requirement is that for each vertex x, the bags that contain x form a consecutive subsequence, $(\mathcal{B}_i, \mathcal{B}_{i+1}, \ldots, \mathcal{B}_j)$, for some i and j with $1 \leq i \leq j \leq n$.

Let G be a graph and let $\mathcal{B} = (\mathcal{B}_i)$ be a path decomposition of G. We define the *width* of $\mathcal{B}$ to be one less than the maximum size of a bag,

$pw(\mathcal{B}) = \max\{|\mathcal{B}_i| - 1\}$, and the *pathwidth* of G to be the minimum width of a path decomposition of G,

$$pw(G) = \min\{pw(\mathcal{B}) : \mathcal{B} \text{ is a path decomposition of } G\}.$$

The pathwidth of a graph has been introduced in [16].

Lemma 1. *Let G be a connected graph with $pw(G) \leq |V_G| - 2$. Then, there is a path decomposition $\mathcal{B}$ of G containing $n \geq 2$ bags such that $pw(\mathcal{B}) = pw(G)$ and, for each $i = 1, \ldots, n-1$, each of $\mathcal{B}_i \setminus \mathcal{B}_{i+1}$, $\mathcal{B}_{i+1} \setminus \mathcal{B}_i$ and $\mathcal{B}_i \cap \mathcal{B}_{i+1}$ is nonempty.*

The pathwidth of a graph can be characterised via a pursuit game on a graph. Rather than describe the cop and robber dynamics of the game, we will simply examine it as an exercise in cleaning a graph. In this game, the cops do not move along the edges of the graph. Each cop has two moves available to it (although at any point in time only one is possible): (1) if a cop is currently on a vertex in the graph, it may be "lifted" off the graph; and (2) if a cop is currently not in the graph, it may be "placed" on any vertex in the graph. On each of the cop's turns, each of his pieces may make only one move – moving a cop from one vertex to another requires two turns. Initially, every edge is marked as dirty (as opposed to the zero-visibility game, where the vertices are the objects being cleaned). An edge is cleaned when both of its endpoints are occupied by a cop. After each move by the cop player, a clean edge is recontaminated if there is a path joining it to a dirty edge which contains no cops – in this game, the robber moves arbitrarily fast.

This pursuit game is often referred to as *node-searching*; it can be shown that $pw(G) \leq k$ if and only if there is a successful node-search strategy on G utilising $k + 1$ cops [7,12].

We introduce this second pursuit game as it is utilised in the proof of Lemma 5; the remainder of this work deals exclusively with the zero-visibility game previously defined.

We produce the following series of inequalities relating the zero-visibility copnumber, the monotonic zero-visibility copnumber and the pathwidth of a graph. In [19], a pursuit game referred to as *strong mixed search* is introduced. It is possible to prove the results in this section utilising the relationships shown therein between strong mixed search and the pathwidth of a graph.

Theorem 4. *Let G be a connected graph containing two or more vertices. Then, $c_0(G) \leq pw(G)$.*

Theorem 5. *Let G be a connected graph; then, $mc_0(G) \leq 2pw(G) + 1$.*

Theorem 6. *Let G be a connected graph; then, $pw(G) \leq 2mc_0(G) - 1$.*

Corollary 1. *Let G be a connected graph on two or more vertices. Then,*

$$c_0(G) \leq pw(G) \leq 2mc_0(G) - 1 \leq 4pw(G) + 1.$$

In Section 4 we provide constructions that in particular prove that the bound in Theorem 4 is tight and the bound in Theorem 5 is tight up to a small additive constant. Moreover, we also use Theorem 3 to argue that the bound in Theorem 6 is tight as well. The formal analysis of these facts is postponed till Section 5.

4 Constructions

We present two constructions of graphs that elicit very interesting relationships between their pathwidth and their zero-visibility copnumbers.

Let G be a graph; the distance between any two vertices x and y is the minimum length of a path joining x and y and is denoted $d_G(x, y)$. So, if $d_G(x, y) = k \geq 1$, then there is a path joining x and y of length k and there are no shorter such paths. If there are no paths joining x and y, the convention is that $d_G(x, y) = \infty$. If H is a subgraph of G, we have $d_H(x, y) \geq d_G(x, y)$ whenever x and y are both present in H. We say that H is an *isometric subgraph* of G if $d_H(x, y) = d_G(x, y)$ whenever x and y are both present in H. Lemma 2 appears in [17].

Lemma 2. *Let G be a graph. If H is an isometric subgraph of G, then $c_0(H) \leq c_0(G)$.*

Let G be a graph; we refer to an edge $e = xy$ as a *cut edge* if the graph $G \setminus e$ obtained by deleting e (without deleting either of the endpoints x or y) is disconnected. Clearly, if e is a cut edge, then $G \setminus e$ has two connected components. Not every graph contains cut edges.

Lemma 3. *Let G be a graph that contains a cut edge e. If H is one of the connected components of $G \setminus e$, then*

$$c_0(H) \leq c_0(G) \text{ and } mc_0(H) \leq mc_0(G).$$

Moreover, let $\mathcal{L}$ be a successful strategy on G. Then, at some point in the strategy at least $c_0(H)$ cops are simultaneously present in H; if $\mathcal{L}$ is a monotonic strategy, at some point at least $mc_0(H)$ cops are simultaneously present in H.

Corollary 2. *Let G be a tree. If H is a subtree of G, then $mc_0(H) \leq mc_0(G)$.*

Example 1. We produce an interesting example of a graph G with an isometric subgraph H such that $mc_0(G) < mc_0(H)$. This illustrates that Lemmas 2 and 3 and Corollary 2 are limited in how they might be extended.

The graph G in question (see Figure 1) contains a large number of vertices with degree 2; to simplify the depiction, some of them are omitted. Specifically, the two dashed lines are paths of length 8 – they each contain 7 internal degree-2 vertices which are not shown. The subgraph H is obtained by deleting the two paths of length 8 (drawn with dashed lines) and all 7 of their internal vertices.

First, we claim that 2 cops can clean the entire graph in a monotonic fashion. One of them is initially placed on y and stays idle during the entire strategy.

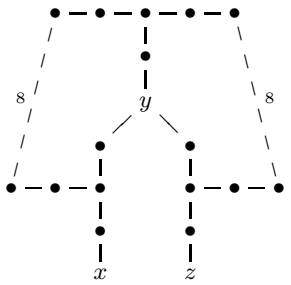


Fig. 1. The subgraph obtained by deleting the two paths of length 8 has strictly higher monotonic zero-visibility copnumber than the supergraph

The other cop is initially placed on x and moves around the remainder of the graph ending on z and cleaning every other vertex. This cop moving around the perimeter cleans the other three vertices adjacent to y – when it reaches a degree-3 vertex it moves onto the neighbour of y and then back.

However, if we delete the two paths drawn as dashed lines, we cannot clean the subgraph monotonically with only two cops – this can be shown in a manner very similar to the proof of Lemma 4.

A *rooted tree* is a tree G where a single vertex has been marked as the *root*. In a rooted tree with root r every vertex $x \neq r$ has a unique parent. The parent of x is identified in the following manner: every vertex $x \neq r$ is joined to r by a unique path – the parent of x is the sole neighbour of x in this path. If y is the parent of x, then x is a child of y; we also use the similarly defined terms grandparent and grandchild when discussing rooted trees.

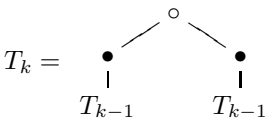
Example 2. The following family of trees illustrates the distinction between the pathwidth of a tree and its monotonic zero-visibility copnumber. Let T_k be obtained by beginning with the full rooted binary tree of height k and subdividing every edge exactly once. We draw the root of each T_k with a circle – see Figure 2 for the first three such trees. By Lemmas 4 and 5, we have

$$mc_0(T_k) = k \text{ and } pw(T_k) = c_0(T_k) = \left\lfloor \frac{k}{2} \right\rfloor + 1.$$

The recursive Algorithm 1 cleans T_k with a monotonic strategy (this can be shown simply via induction) that utilises $k + 1$ cops and begins with every cop placed on the root. However, this is not an optimal strategy – if $k \geq 1$, there is a successful monotonic strategy on T_k using k cops. We obtain this strategy by using Algorithm 1 as follows. We first express T_k as two copies of T_{k-1} joined by a path of length 4:

Algorithm 1. CLEAN(T_k)

Require: There are $k+1$ cops on the root of the graph T_k.
if $k = 0$ **then**
return
end if
1. Let x be the root of T_k and let y and z be the two grandchildren of x. Let T^y and T^z be the two copies of T_{k-1} rooted at y and z, respectively.
2. In two moves, move $k-1$ of the cops from the root of T_k to y, leaving the remaining cop on x.
3. CLEAN(T^y).
4. Move all $k-1$ cops from T^y to z (each cop does not move any further into T^z).
5. CLEAN(T^z).
return



We clean one copy of T_{k-1} by using Algorithm 1 in such a way that one cop remains on the root of this subtree during the entire strategy. All k cops then move to the root of the other copy of T_{k-1} and clean that subtree in a similar fashion. Thus, $mc_0(T_k) \leq k$, if $k \geq 1$. In Lemma 4, we show that this strategy is, in fact, optimal.

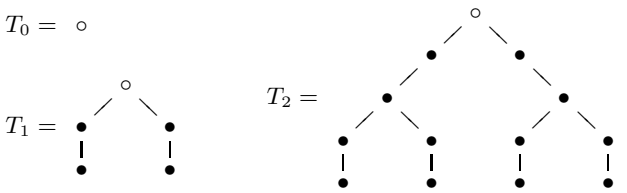


Fig. 2. The first three subdivided binary trees

Lemma 4. *For $k \geq 1$, $mc_0(T_k) = k$.*

Theorem 4 and Lemma 4 together imply that $pw(T_k) \leq 2k-1$. However, we in fact have $pw(T_k) = \lfloor \frac{k}{2} \rfloor + 1$.

Lemma 5. *For $k \geq 1$, $pw(T_k) = c_0(T_k) = \lfloor \frac{k}{2} \rfloor + 1$.*

We produce a construction of graphs with $c_0 < pw$ and $c_0 < mc_0$ with the unbounded ratios in both inequalities.

A *universal vertex* in a graph G is a vertex adjacent to every other vertex. Given a graph G, we form the graph G^* by adding a universal vertex to G – that is, a single new vertex is added together with edges joining this new vertex and every other vertex already present in G.

A *subdivision* of a graph G is a graph H formed by replacing one or more edges in G with paths of length greater than or equal to 2; a subdivision is formed by dividing an edge into two or more new edges. If the vertices of G are labeled and H is a subdivision of G, we preserve the labeling of the vertices, adding new labels to the new vertices.

Lemma 6. *If G is a tree containing two or more vertices, then there is a subdivision H of G such that $c_0(H^*) = 2$.*

5 Comparisons between the Zero-Visibility Copnumbers and the Pathwidth of a Graph

We examine in detail the inequality presented in Corollary 1:

$$c_0(G) \leq pw(G) \leq 2mc_0(G) - 1 \leq 4pw(G) + 1.$$

A *caterpillar* is a tree such that deleting all vertices of degree 1 results in a path or an empty graph. The proof of Theorem 7 is a straightforward exercise and is omitted.

Theorem 7. *Let G be a graph. The following are equivalent:*

1. *We have $c_0(G) = 1$, $mc_0(G) = 1$ or $pw(G) = 1$.*
2. *We have $c_0(G) = mc_0(G) = pw(G) = 1$.*
3. *We have $c_0(G) = pw(G) = 2mc_0(G) - 1$.*
4. *The graph G is a caterpillar.*

The class of graphs that minimizes the zero-visibility copnumbers is identical to the class that minimises pathwidth. However, if $c_0(G) = 2$, this gives us absolutely no information concerning $mc_0(G)$ or $pw(G)$, as we will see in Theorem 8.

Remark 1. The bound $c_0(G) \leq pw(G)$ in Theorem 4 is the best possible.

The class of graphs $\{T_k\}$, described in the previous section, satisfy $c_0(T_k) = pw(T_k) < mc_0(T_k)$ (by Lemmas 4 and 5). Thus, the bound $c_0(G) \leq pw(G)$ is tight on an infinite family of graphs and we cannot sandwich mc_0 between c_0 and pw, in general.

Remark 2. The bound $mc_0(G) \leq 2pw(G)+1$ in Theorem 5 can only be improved by a small constant, if it can be improved at all.

The subdivided binary trees T_k described in Example 2 have $mc_0(T_k) = k$ and $pw(T_k) = \lfloor \frac{k}{2} \rfloor + 1$. So,

$$mc_0(T_k) = \begin{cases} 2pw(T_k) - 1 \text{ if } k \text{ is odd, or} \\ 2pw(T_k) - 2 \text{ if } k \text{ is even.} \end{cases}$$

The coefficient of 2 cannot be reduced; only the additive constant can be changed, possibly by reducing it by one or two. In a roundabout manner, this shows that the result in [4] is close to optimal.

Remark 3. The bound $pw(G) \leq 2mc_0(G) - 1$ in Theorem 6 is the best possible.

The complete graph on an even number of vertices has

$$pw(K_{2m}) = 2mc_0(K_{2m}) - 1 = 2m - 1.$$

As well, Theorem 8 shows that we cannot replace Theorem 6 with the stronger statement "$pw(G) \leq 2c_0(G) - 1$". In fact, pathwidth cannot be bounded above by any function of the zero-visibility copnumber.

Theorem 8. *For any positive integer k, there is a graph G with $c_0(G) = 2$ and $pw(G) \geq k$.*

6 Conclusion

There remains a considerable amount of further work concerning the zero-visibility model to be accomplished. Characterisations of c_0 and mc_0 over well-known families of graphs (such as trees, unicyclic graphs, planar graphs, series parallel graphs, *etc.*) are of interest. An analysis of the algorithmic complexity of accomplishing a successful zero-visibility search would cement this model's position in the overall area of pursuit games and width parameters. It would be very interesting to construct some sort of relationship between the value $mc_0(G) - c_0(G)$ (or possibly $mc_0(G)/c_0(G)$) and combinatoric or connective properties of the graph – that is, to answer the question, given some known property of the graph, can we bound the amount by which c_0 and mc_0 differ?

The fact that the monotonic zero-visibility copnumber can be bounded both above and below by positive multiples of the pathwidth suggests that, in a sense, node-search and the monotonic zero-visibility search are variations of the same game – each number is an approximation of the other, suggesting that efficient strategies in one game can usually be translated to efficient strategies in the other.

However, Theorem 8 shows that the zero-visibility copnumber can be entirely unrelated to the pathwidth and the monotonic zero-visibility copnumber. The general zero-visibility search can be carried out using methods that will not work in a node-search – the zero-visibility search is genuinely distinct from other pursuit games and informs us of different structural properties of a graph.

References

1. Adler, M., Räcke, H., Sivadasan, N., Sohler, C., Vöcking, B.: Randomized pursuit-evasion in graphs. Combinatorics, Probability and Computing 12, 225–244 (2003)
2. Bienstock, D., Seymour, P.: Monotonicity in graph searching. Journal of Algorithms 12, 239–245 (1991)
3. Clarke, N.E., MacGillivray, G.: Characterizations of k-copwin graphs. Discrete Mathematics 312, 1421–1425 (2012)
4. Dereniowski, D.: From pathwidth to connected pathwidth. SIAM Journal on Discrete Mathematics 26, 1709–1732 (2012)
5. Dereniowski, D., Dyer, D., Tifenbach, R., Yang, B.: The zero-visibility copnumber of a tree (2013)
6. Edmonds, J.: Paths, trees, and flowers. Canadian Journal of Mathematics 17, 449–467 (1965)
7. Ellis, J.A., Sudborough, I.H., Turner, J.S.: The vertex separation and search number of a graph. Information and Computing 113, 50–79 (1994)
8. Isler, V., Kannan, S., Khanna, S.: Randomized pursuit-evasion with local visibility. SIAM Journal on Discrete Mathematics 20, 26–41 (2006)
9. Isler, V., Karnad, N.: The role of information in the cop-robber game. Theoretical Computer Science 399, 179–190 (2008)
10. Jeliazkova, D.: Aspects of the cops and robber game played with incomplete information. Master's thesis, Acadia University (2006)
11. Kehagias, A., Mitche, D., Prałat, P.: The role of visibility in the cops-robber game and robotic pursuit/evasion (2012)
12. Kinnersley, N.G.: The vertex separation number of a graph equals its path-width. Information Processing Letters 42, 345–350 (1992)
13. LaPaugh, A.S.: Recontamination does not help to search a graph. Journal of the ACM 40, 224–245 (1993)
14. Nowakowski, R., Winkler, P.: Vertex-to-vertex pursuit in a graph. Discrete Mathematics 43, 235–239 (1983)
15. Quilliot, A.: Problèmes de jeux, de point fixe, de connectivité et de représentation sur des graphes, des ensembles ordonnés et des hypergraphes. PhD thesis, Université de Paris VI (1983)
16. Robertson, N., Seymour, P.D.: Graph minors. I. Excluding a forest. Journal of Combinatorial Theory, Series B 35, 39–61 (1983)
17. Tang, A.: Cops and robber with bounded visibility. Master's thesis, Dalhousie University (2004)
18. Tošić, R.: Inductive classes of graphs. In: Proceedings of the Sixth Yugoslav Seminar on Graph Theory, pp. 233–237. University of Novi Sad (1985)
19. Yang, B.: Strong-mixed searching and pathwidth. Journal of Combinatorial Optimization 13, 47–59 (2007)
20. Yang, B., Dyer, D., Alspach, B.: Sweeping graphs with large clique number. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 908–920. Springer, Heidelberg (2004)
21. Yang, B., Dyer, D., Alspach, B.: Sweeping graphs with large clique number. Discrete Mathematics 309, 5770–5780 (2009)

On (k, ℓ)-Graph Sandwich Problems

Fernanda Couto[1], Luérbio Faria[2], Sulamita Klein[1],
Fábio Protti[3], and Loana T. Nogueira[3]

[1] UFRJ, Rio de Janeiro, Brazil
[2] UERJ, Rio de Janeiro, Brazil
[3] UFF, Rio de Janeiro, Brazil
{nandavdc,sula,luerbio}@cos.ufrj.br, {fabio,loana}@ic.uff.br

Abstract. In this work we consider the Golumbic, Kaplan and Shamir graph sandwich decision problem for property Π, where given two graphs $G^1 = (V, E^1)$ and $G^2 = (V, E^2)$, the question is to know whether there exists a graph $G = (V, E)$ such that $E^1 \subseteq E \subseteq E^2$ and G satisfies property Π. The main property Π we are interested in this paper is "being a (k, ℓ)-graph". We say that a graph $G = (V, E)$ is (k, ℓ) if there is a partition of the vertex set of G into at most k independent sets and at most ℓ cliques. We prove that the STRONGLY CHORDAL-$(2, \ell)$ GRAPH SANDWICH PROBLEM is NP-complete, for $\ell \geq 1$, and that the CHORDAL-(k, ℓ) GRAPH SANDWICH PROBLEM is NP-complete, for $k \geq 2$, $\ell \geq 1$. We also introduce in this paper a new work proposal related to graph sandwich problems: the GRAPH SANDWICH PROBLEM WITH BOUNDARY CONDITIONS. Our goal is to redefine well-known NP-complete graph sandwich problems by cleverly assigning properties to its input graphs so that the redefined problems are polynomially solvable. Let POLY-COLOR(k) denote an infinite family of graphs G for which deciding whether G is k-colorable can be done in $O(p(n))$ time, where p is a polynomial and $n = |V(G)|$. In order to illustrate how boundary conditions can change the complexity status of a graph sandwich problem, we present here a polynomial-time solution for the (k, ℓ)-GRAPH SANDWICH PROBLEM FOR ALL k, ℓ, when beforehand we know that G^1 belongs to POLY-COLOR(k) and G^2 has a polynomial number maximal of cliques.

Keywords: Graph Sandwich Problems, Boundary Conditions, (k, ℓ)-Graphs, Strongly Chordal-$(2, \ell)$ Graphs, Strongly Chordal-$(2, 1)$ Graphs, Chordal-$(2, 1)$ Graphs.

1 Introduction

In 1995, Golumbic, Kaplan and Shamir [12] introduced the SANDWICH PROBLEM:

GRAPH SANDWICH PROBLEM FOR PROPERTY Π ($\Pi-$SP)
Instance: $G^1 = (V, E^1)$ and $G^2 = (V, E^2)$, such that $E^1 \subseteq E^2$.
Question: Is there a graph $G = (V, E)$ such that $E^1 \subseteq E \subseteq E^2$ and G satisfies property Π?

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 187–197, 2013.

Observe that the graph G, if it exists, must be "sandwiched" between the graphs G^1 and G^2 and must satisfy property Π. In order to avoid a trivial problem, we assume that G^1 and G^2 do not satisfy property Π.

Given two graphs $G^1 = (V, E^1)$ and $G^2 = (V, E^2)$ with the same vertex set V and $E^1 \subseteq E^2$, a graph $G = (V, E)$ is called a *sandwich* graph for the pair G^1, G^2 if $E^1 \subseteq E \subseteq E^2$. Let $E^3 = E(\overline{G^2})$ denote the set of *forbidden* edges. We call E^1 the *mandatory* edge set, and $E^2 \setminus E^1$ the *optional* edge set. Hence, any sandwich graph $G = (V, E)$ for the pair G^1, G^2 must contain all mandatory edges and no forbidden edge.

The *recognition* problem for a class of graphs $\mathcal{C}$ is equivalent to the particular graph sandwich problem where $E^1 = E^2$, that is, the optional edge set is empty. Graph sandwich problems have attracted much attention lately because of many applications and by the fact that they naturally generalize recognition problems [5–7, 11, 17]. Note that Π−SP is clearly at least as hard as the problem of recognizing graphs with property Π, since given a polynomial-time algorithm for Π−SP, it is possible to use this algorithm with $E^1 = E^2 = E$ to recognize if a graph $G = (V, E)$ satisfies property Π. Thus, in such cases, we have problems in P that cannot be easier when regarded as sandwich problems.

A graph G is (k, ℓ) if $V(G)$ can be partitioned into k independent sets and ℓ cliques (a (k, ℓ)-partition). The main property Π we are interested in this paper is "being a (k, ℓ)-graph ". The problem of recognizing if a graph G is (k, ℓ) was shown to be NP-complete if $k \geq 3$ or $\ell \geq 3$ and solvable in polynomial time otherwise [1–3]. In [1, 3, 9], polynomial-time algorithms have been developed for deciding if a graph admits a $(2, 1)$ or a $(2, 2)$-partition.

Let ℓ and k be two non-negative integers. We denote by $(\ell+1)K_{k+1}$ the graph obtained from the disjoint union of $(\ell+1)$ copies of K_{k+1}. Chordal-(k, ℓ) graphs have been well studied in the literature. Hell et al. [10, 13, 14] have analyzed algorithmic and complexity aspects of chordal-(k, ℓ) graphs, proving that the recognition problems of chordal-(k, ℓ) and strongly chordal-(k, ℓ) graphs are in P, using the following characterization:

Theorem 1. [13] *A chordal graph G is (k, ℓ) if and only if G does not contain a $(\ell + 1)K_{k+1}$ as an induced subgraph.*

In [12], Golumbic et al. have presented a diagram showing the complexity status (at that time) of the sandwich problem for some subfamilies of perfect graphs. We can notice that the majority of the studied problems are NP-complete. Since then, many other problems, like $(2, 1)$-SP [4] and STRONGLY CHORDAL-SP [11], have been proved to be NP-complete.

These two last problems have inspired us to question about the complexity of the STRONGLY CHORDAL-$(2, 1)$-SP problem, since the recognition of strongly chordal-(k, ℓ) graphs can be done in polynomial time [10, 13, 14]. In this work we prove that STRONGLY CHORDAL-$(2, 1)$-SP is NP-complete. Based on this result, we prove that the STRONGLY CHORDAL-$(2, \ell)$ GRAPH SANDWICH PROBLEM is NP-complete, for $\ell \geq 1$, and that the CHORDAL-(k, ℓ) GRAPH SANDWICH PROBLEM is NP-complete, for $k \geq 2,\ \ell \geq 1$.

Due to the difficulty of graph sandwich problems, we started thinking about some properties that, when cleverly applied, could turn NP-complete sandwich problems into polynomially solvable ones. In this paper we introduce this new work proposal: we consider special properties for the input graphs of a graph sandwich problem, and therefore define a generalized version of sandwich problems, called SANDWICH PROBLEM WITH BOUNDARY CONDITIONS. Note that, in contrast to Golumbic et al.'s problem, what makes sense in this case is dealing with difficult problems that, when treated with boundary conditions, cannot be more difficult. We have considered a particular NP-complete sandwich problem: the (k, ℓ)-SANDWICH PROBLEM, for $k + \ell > 2$. We prove in this work that by adding some boundary conditions we can turn it into a polynomially solvable problem for all k, ℓ.

2 The Strongly Chordal-$(2, 1)$ Graph Sandwich Problem

A graph $G = (V, E)$ is *chordal* if every cycle of length at least four contains a chord, i.e, an edge between two nonconsecutive vertices. It is well known [15, 18] that chordal graphs can be recognized in polynomial time. The CHORDAL GRAPH SANDWICH PROBLEM was shown to be NP-complete in the fundamental sandwich problem paper [12].

A *sun* is a chordal graph on $2n$ vertices ($n \geq 3$) whose vertex set can be partitioned into $W = \{w_1, \ldots, w_n\}$ and $U = \{u_1 \ldots, u_n\}$ such that W is an independent set and u_i is adjacent to w_j if and only if $i = j$ or $i = j + 1$ (mod n). *Strongly chordal graphs* were defined by Farber [8] as chordal graphs that do not contain a sun. He also proved that strongly chordal graphs can be recognized in polynomial time. The STRONGLY CHORDAL GRAPH SANDWICH PROBLEM was shown to be NP-complete in [11]. But what happens if we ask for a sandwich graph G being strongly chordal graph and also a $(2, 1)$-graph? The problem can be formulated as follows:

STRONGLY CHORDAL-$(2, 1)$ GRAPH SANDWICH PROBLEM
Instance: $G^1 = (V, E^1)$ and $G^2 = (V, E^2)$, such that $E^1 \subseteq E^2$.
Question: Is there a graph $G = (V, E)$ such that $E^1 \subseteq E \subseteq E^2$ and G is a strongly chordal-$(2, 1)$ graph?

We prove that the STRONGLY CHORDAL-$(2, 1)$ SANDWICH PROBLEM is NP-complete by showing a reduction from the NP-complete problem STRONGLY TRIANGULATING A COLORED GRAPH (STCG).

This decision problem is defined below.

STRONGLY TRIANGULATING A COLORED GRAPH (STCG)
Instance: Graph $G = (V, E)$, proper vertex coloring $c : V \to Z$.
Question: Does there exist a supergraph $G' = (V, E')$ of G that is strongly chordal and also properly colored by c?

Theorem 2. *The* STRONGLY CHORDAL-$(2,1)$ GRAPH SANDWICH PROBLEM *is NP-complete.*

Proof. The problem is in NP, since given a graph G that is a solution of the problem (a yes-certificate), we can check in polynomial time if it is a strongly chordal graph [8] (in this case also chordal), a chordal-$(2,1)$ graph [16], a supergraph of G^1, and a subgraph of G^2. In order to reduce STCG to the STRONGLY CHORDAL-$(2,1)$ GRAPH SANDWICH PROBLEM we will use the instance (V, E^1, E^2) used to prove that the STRONGLY CHORDAL GRAPH SANDWICH PROBLEM is NP-complete (see [11]) and show that its vertex set can be partitioned into two independent sets and one clique, i.e. G is a $(2,1)$-graph. The details of this proof can be found in the appendix.

Corollary 1. *The* CHORDAL-$(2,1)$ GRAPH SANDWICH PROBLEM *is NP-complete.*

2.1 The Strongly Chordal-(k,ℓ) Graph Sandwich Problem, $k \geq 2, \ell \geq 1$

After solving the STRONGLY CHORDAL-$(2,1)$ GRAPH SANDWICH PROBLEM, the question about the complexity of the STRONGLY CHORDAL-(k,ℓ) GRAPH SANDWICH PROBLEM, for $k \geq 2, \ell \geq 1$, naturally arises. We have succeeded to prove that the STRONGLY CHORDAL-$(2,\ell)$ GRAPH SANDWICH PROBLEM, for $\ell \geq 1$, is NP-complete. This problem is formulated as follows:

STRONGLY CHORDAL-$(2,\ell)$ GRAPH SANDWICH PROBLEM, $\ell \geq 1$
Instance: $G^1 = (V, E^1)$ and $G^2 = (V, E^2)$, such that $E^1 \subseteq E^2$.
Question: Is there a graph $G = (V, E)$ such that $E^1 \subseteq E \subseteq E^2$ and G is a strongly chordal-$(2,\ell)$ graph, $\ell \geq 1$?

Lemma 1. *Given $k \geq 2$ and $\ell \geq 1$, if* STRONGLY CHORDAL-(k,ℓ)-SP *is NP-complete, then* STRONGLY CHORDAL-$(k,\ell+1)$-SP *is NP-complete.*

Proof. We remark that the STRONGLY CHORDAL-(k,ℓ) GRAPH SANDWICH PROBLEM, $k \geq 2, \ell \geq 1$ is in NP, since we can check in polynomial time whether a graph G is a sandwich for a pair (G^1, G^2) and whether it is strongly chordal-(k,ℓ) [10, 13, 14].

We consider the following special instance $(G^{1'}, G^{2'})$ of STRONGLY CHORDAL-$(k,\ell+1)$-SP obtained from (G^1, G^2), an instance of the NP-complete problem STRONGLY CHORDAL-(k,ℓ)-SP, such that there is a (k,ℓ)-strongly chordal sandwich graph G for (G^1, G^2) if and only if there is a strongly chordal-$(k,\ell+1)$ sandwich graph G', $k \geq 2$, $\ell \geq 1$, for $(G^{1'}, G^{2'})$.

From (G^1, G^2) we define one additional clique K, such that $|K| = k+1$. We set $V(G^{1'}) = V(G^{2'}) = V(G^1) \cup V(K)$, $E(G^{1'}) = E(G^1) \cup E(K)$, and $E(G^{2'}) = E(G^2) \cup E(K)$. This concludes the construction of the instance $(G^{1'}, G^{2'})$.

Suppose there is a strongly chordal-(k,ℓ) sandwich graph G for (G^1, G^2). Consider G' formed by G plus the forced edges of $(G^{1'}, G^{2'})$. In order to prove

that graph G' is a strongly chordal graph we consider the strong elimination sequence started by any sequence of the vertices of K, followed by a strong elimination sequence of the strongly chordal graph G. In order to prove that G' is $(k, \ell+1), k \geq 2, \ell \geq 1$, we consider a (k, ℓ)-partition for G, and we set a $(k, \ell+1)$-partition for G' formed by the k independent sets, the ℓ cliques of G, and K.

Suppose there is a strongly chordal-$(k, \ell+1)$ sandwich graph G' for $(G^{1'}, G^{2'})$, $k \geq 2, \ell \geq 1$. Given $G = G' - K$, we will prove that G is a strongly chordal-(k, ℓ)-sandwich graph for (G^1, G^2). Suppose by contradiction that G is not a strongly chordal-(k, ℓ) graph. First, note that, since being a strongly chordal graph is an hereditary property, G must be strongly chordal. Thus, if G is not strongly chordal-(k, ℓ) then it is because G is not a (k, ℓ)-graph. It follows from Theorem 1 that G contains a $(\ell+1)(K_{k+1})$ as an induced subgraph. Since G' is the disjoint union of G and K, there is an induced subgraph $(\ell+2)K_{k+1}$ of G' formed by K and the induced subgraph $(\ell+1)(K_{k+1})$ of G. By Theorem 1, G' is not strongly chordal-$(k, \ell+1)$, a contradiction. Hence, G is a strongly chordal-(k, ℓ) graph.

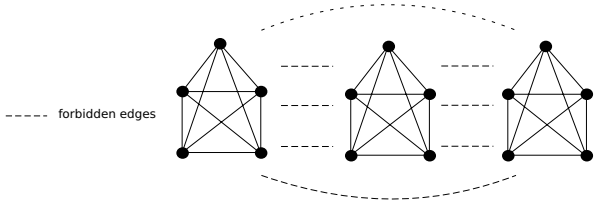


Fig. 1. Example of a $3K_5$, the forbidden induced subgraph for chordal-$(4, 2)$ graphs

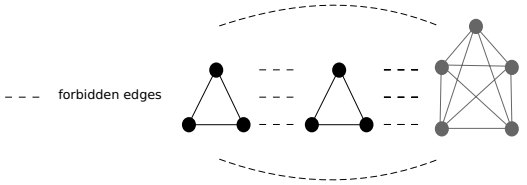


Fig. 2. Example of the instance when $k = 2$ and $\ell = 1$. Note that when G has two isolated triangles, G' will have three isolated triangles.

Theorem 3. *The* STRONGLY CHORDAL-$(2, \ell)$ GRAPH SANDWICH PROBLEM*, for* $\ell \geq 1$*, is NP-complete.*

Proof. The proof of Theorem 3 is done by induction using Theorem 2 and Lemma 1.

We remark that the next two results are on chordal graphs.

Lemma 2. *Given* $k \geq 2$ *and* $\ell \geq 1$*, if* CHORDAL-(k, ℓ)-SP *is NP-complete, then* CHORDAL-$(k+1, \ell)$-SP *is NP-complete.*

Proof. Observe that the CHORDAL-(k,ℓ) GRAPH SANDWICH PROBLEM, $k \geq 2$, $\ell \geq 1$, is in NP, since we can check in polynomial time whether a graph G is a sandwich for a pair (G^1, G^2) and whether it is chordal-(k,ℓ) [13].

We consider the following special instance $(G^{1'}, G^{2'})$ of the problem CHORDAL-$(k+1,\ell)$-SP obtained from (G^1, G^2), an instance of the NP-complete problem CHORDAL-(k,ℓ)-SP, such that there is a (k,ℓ)-chordal sandwich graph G for (G^1, G^2) if and only if there is a $(k+1,\ell)$-chordal sandwich graph G', $k \geq 2$, $\ell \geq 1$ for $(G^{1'}, G^{2'})$.

We denote by n_1 and n_2, respectively, the number of K_{k+1}'s of G^1, and G^2. Let $K^1_{k+1}, K^2_{k+1}, \cdots, K^{n_2}_{k+1}$ be the subgraphs of G^2 isomorphic to K_{k+1}. For each K^i_{k+1} of G^2 we define one additional vertex u_i, such that $V(K^i_{k+1}) \cup \{u_i\}$ forms a clique K^i, $i = 1, \cdots, n_2$. We set $V(G^{1'}) = V(G^{2'}) = V(G^1) \bigcup_{i=1}^{n_2} \{u_i\}$, $E(G^{1'}) = E(G^1) \bigcup_{i=1}^{n_1} E(K^i)$, and $E(G^{2'}) = E(G^2) \bigcup_{i=1}^{n_2} E(K^i)$. This concludes the construction of the instance $(G^{1'}, G^{2'})$.

Suppose there is a chordal-(k,ℓ) sandwich graph G for (G^1, G^2). Consider G' formed by G plus the forced edges of $(G^{1'}, G^{2'})$. In order to prove that G' is a chordal graph we consider the perfect elimination sequence started by the simplicial vertices u_i, $i = 1, \cdots, n_2$, and followed by a perfect elimination sequence of the chordal graph G. In order to prove that G' is a $(k+1,\ell)$-graph, $k \geq 2$, $\ell \geq 1$, we consider a (k,ℓ)-partition of G, and we define a $(k+1,\ell)$-partition for G' formed by the k independents sets of G, the independent set $\bigcup_{i=1}^{n_2} \{u_i\}$, and the ℓ cliques of G.

Suppose there is a chordal-$(k+1,\ell)$ sandwich graph G' for $(G^{1'}, G^{2'})$, $k \geq 2$, $\ell \geq 1$. We consider $G = G' - \bigcup_{i=1}^{n_2} \{u_i\}$ and we will prove that G is a chordal-(k,ℓ)-sandwich graph for (G^1, G^2). Suppose by contradiction that G is not a chordal-(k,ℓ) graph. Note that, since being a chordal graph is an hereditary property, if G is not chordal then G' is not chordal either. It follows from Theorem 1 that G contains a $(\ell+1)K_{k+1}$ as an induced subgraph (see Figure 1). By construction of G', for each K^i_{k+1} of G^2 there is an additional vertex u_i forming a K_{k+2} in $G^{2'}$ (See Figure 3 for an example). Then, we have $(\ell+1)$ copies of K_{k+2} in G' which, by the characterization in [13], implies that G' is not chordal-$(k+1,\ell)$. Hence, G must be chordal-(k,ℓ).

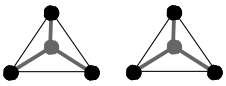


Fig. 3. Example of the addition of vertices u_i when $k = 2$ and $\ell = 1$. Note that when G has two isolated triangles, G' will have two isolated K_4's.

Theorem 4. *The* CHORDAL-(k,ℓ) GRAPH SANDWICH PROBLEM*, for $k \geq 2$ and $\ell \geq 1$, is NP-complete.*

Proof. The proof of Theorem 4 is done by a 2-step mathematical induction on k and ℓ where the induction base follows from Corollary 1 and the inductive steps follow from Lemma 1 and Lemma 2.

3 Graph Sandwich Problems with Boundary Conditions

We have noticed that the majority of the known sandwich problems are NP-complete. The work by Golumbic et al. [12] contains a diagram showing the complexity status (at that time) of the sandwich problem for some subfamilies of perfect graphs. Other papers are also worth mentioning [4, 11].

Due to the difficulty of graph sandwich problems, we started thinking about some properties that, when cleverly applied, could turn NP-complete sandwich problems into polynomially solvable ones. In this paper we propose another way of dealing with sandwich problems: instead of choosing just the property Π, we will choose *boundary conditions*, i.e, properties Π^i assigned to the input graphs $G^i, i = 1, 2$, in order to try polynomial-time solutions for such modified versions of known NP-complete sandwich problems. We can formulate this version as follows:

GRAPH SANDWICH PROBLEM FOR PROPERTY Π WITH BOUNDARY CONDITIONS
Instance: $G^1 = (V, E^1)$ satisfying Π^1 and $G^2 = (V, E^2)$ satisfying Π^2, such that $E^1 \subseteq E^2$.
Question: Is there a graph $G = (V, E)$ such that $E^1 \subseteq E \subseteq E^2$ and G satisfies the property Π?

We denote this problem by a triple (Π^1, Π, Π^2)-SP. When we do not require that G^i satisfies a property Π^i we denote Π^i by $*$. Note that the GRAPH SANDWICH PROBLEM WITH BOUNDARY CONDITIONS generalize the GRAPH SANDWICH PROBLEM.

In this section we focus on a particular graph class, the class of (k, ℓ)-graphs [1], related to partition problems [9]. The recognition problem for this class is NP-complete for $k, \ell \geq 3$ [1–3], and polynomially solvable otherwise. Consequently, we know that the sandwich problem is trivially NP-complete for $k, \ell \geq 3$. In [4], the authors prove that the problem remains NP-complete for $k + \ell \geq 3$.

3.1 (Poly-Color(k), (k, ℓ), Polynomial Number of Maximal Cliques)-SP

It is interesting to see how the status of a problem can change from NP-complete to polynomially solvable when we assign to G^1 or G^2 some particular conditions. We will study the (k, ℓ)-GRAPH SANDWICH PROBLEM with some strong conditions, that will allow a polynomial-time solution for some related sandwich problems. We do not consider the case when G^1 or G^2 satisfies property Π, since they are trivial and can be solved by setting $G = G^1$ or $G = G^2$.

Now we give some useful definitions that will be used in the rest of the text.

A graph is *k-colorable* if its vertices can be properly colored (i.e. adjacent vertices do not receive the same color) with at most k colors.

Definition 1. *For a fixed k,* Poly-color(k) *denotes an infinite family of graphs G for which there exists a polynomial p such that deciding wheter G is k-colorable can be done in time $O(p(n))$, where $n = |V(G)|$.*

Definition 2. POLYNOMIAL NUMBER OF MAXIMAL CLIQUES, *or simply* PNMC, *denotes an infinite family of graphs G for which there exists a polynomial q such that the number of maximal cliques of G is bounded by $O(q(n))$, where $n = |V(G)|$.*

As an example, the notation POLY-COLOR(k) can be used to denote chordal graphs. PNMC can also stand for chordal graphs, since the number of maximal cliques of a chordal graph is at most $n - 1$. Our goal in this section is to prove that the following general sandwich problem with boundary conditions is polynomially solvable for all k, ℓ:

(POLY-COLOR(k), (k, ℓ), PNMC)-SP
Instance: A graph $G^1 = (V, E^1)$ in POLY-COLOR(k) and a graph $G^2 = (V, E^2)$ in PNMC such that $E^1 \subseteq E^2$.
Question: Does there exist a (k, ℓ)-graph $G = (V, E)$ such that $E^1 \subseteq E \subseteq E^2$?

Theorem 5. *For fixed k, ℓ,* (POLY-COLOR(k), (k, ℓ), PNMC)-SP *is in P.*

The proof of Theorem 5 is based on Algorithm 1.

```
Algorithm for (POLY-COLOR(k), (k,l), PNMC)-SP
  Input: G^1 = (V,E^1) in POLY_COLOR(k);
         G^2=(V, E^2) in PNMC;
  Output: G=(V,E) a (k,l)-graph, if it exists;

  Let C be the collection of maximal cliques of G^2 ;
  For each subcollection {C_1, C_2, ... , C_l} of C do
      Let C' = V(C_1) U V(C_2) U ... U V(C_l)$;
      If  G^1 \ C' is k-colorable then
          Return {G=(V, E^1 U E(C^1) U ... U E(C_l))};
      Return {there is no (k,l)-graph G such that E^1 <= E(G) <= E^2};
```

Algorithm 1 : solves (POLY-COLOR(k), (k, ℓ), PNMC)-SP, for all k, ℓ

Proof. Assume that the number of maximal cliques of G^2 is at least ℓ, otherwise the theorem is trivially true.

The proof is based on Algorithm 1. First, we show that it runs in polynomial time. Since G^2 has a polynomial number of maximal cliques, we can list them in polynomial time by using, for instance, the algorithm in [19] (which has polynomial delay). Next, since ℓ is fixed, all possible subcollections with ℓ maximal cliques can also be listed in polynomial time. For each one of them, we compute C' and $G^1 \backslash C'$ as shown in the algorithm. Note that $G^1 \backslash C'$ is in POLY-COLOR(k), since it is an induced subgraph of G^1. Therefore, we can test if $G^1 \backslash C'$ is k-colorable in polynomial time (recall that k is fixed as well).

Now, assume that Algorithm 1 returns a positive answer for the problem. Then there exists a subcollection $\{C_1, \ldots, C_\ell\}$ with exactly ℓ maximal cliques of G^2 such that $G^1 \backslash C'$, where $C' = \cup_{i=1}^{\ell} V(C_i)$, is k-colorable. Then it is clear that the

graph $G = (V, E_1 \cup E(C_1) \cup \cdots \cup E(C_\ell))$ returned by the algorithm is a sandwich graph and also a (k,ℓ)-graph: a (k,ℓ)-partition of G is formed by k stable sets of $G^1 \backslash C'$ plus the ℓ cliques C_1, $C_2 \backslash C_1$, $C_3 \backslash (C_1 \cup C_2), \ldots, C_\ell \backslash (C_1 \cup C_2 \cup \cdots \cup C_{\ell-1})$. We remark that, according to the definition of a (k,ℓ)-partition, some parts may be empty.

Finally, assume that Algorithm 1 returns a negative answer for the problem. Assume also by contradiction that there exists a (k,ℓ)-sandwich graph G with a (k,ℓ)-partition formed by stable sets $S_1, \ldots, S_k$ and cliques $Q_1, \ldots, Q_k$. Since G^2 is a sandwich graph, every Q_i is a (not necessarily maximal) clique of G^2. Let Q'_i be a maximal clique of G^2 such that $Q_i \subseteq Q'_i$, $i = 1 \ldots \ell$. If $Q'_i = Q'_j$ for some $i \neq j$, discard one of them, and repeat this process until no duplicates exist. If less than ℓ maximal cliques remain, complete the current subcollection of maximal cliques so that it contains ℓ distinct elements $C_1, C_2, \ldots, C_\ell$ (recall that the number of maximal cliques of G^2 is assumed to be at least ℓ). Then it is clear that the algorithm considers the subcollection $\{C_1, C_2, \ldots, C_\ell\}$. Let $C' = \cup_{i=1}^{\ell} V(C_i)$. Since by construction C' contains $Q = \cup_{i=1}^{\ell} V(Q_i)$, it is clear that $G^1 \backslash C'$ is k-colorable, since it is a subgraph of the k-colorable subgraph $G \backslash Q$. This means that Algorithm 1 returns a positive answer, a contradiction.

We remark that Theorem 5 is a framework which allows us to establish several corollaries as we state next.

Corollary 2. (CHORDAL, (k,ℓ)), CHORDAL)-SP *is solvable in polynomial time.*

Corollary 3. (COGRAPH, (k,ℓ), BOUNDED DEGREE Δ)-SP *is solvable in polynomial time.*

Corollary 4. (COMPARABILITY, (k,ℓ), TRIANGLE FREE)-SP *is solvable in polynomial time.*

4 Conclusion

We have proved that the STRONGLY CHORDAL-$(2,1)$ GRAPH SANDWICH PROBLEM is NP-complete. As a Corollary we have that the CHORDAL-$(2,1)$ GRAPH SANDWICH PROBLEM is NP-complete as well. We also have succeeded to prove that the following problems are NP-complete: STRONGLY CHORDAL-$(2,\ell)$-SP for $\ell \geq 1$ and CHORDAL-(k,ℓ)-SP for $k \geq 2$, $\ell \geq 1$. In Figure 4 we depict our contribution to Golumbic, Kaplan and Shamir's diagram [12] presenting the results in this paper, and some additional results in the literature. We have also introduced a new proposal while working with sandwich problems, assigning special properties to the input graphs G^1 and G^2 in order to try polynomially solvable versions of some well-known NP-complete graph sandwich problems. We call the general version SANDWICH PROBLEM WITH BOUNDARY CONDITIONS. As an application of this strategy, we have presented the general problem (POLY-COLOR(k), (k,ℓ), PNMC)-SP that is polynomial solvable for all k,ℓ, in contrast with (k,ℓ)-SP, $k+\ell \geq 3$, which is NP-complete. Table 1 summarizes the results presented in this paper and the corresponding results in the literature.

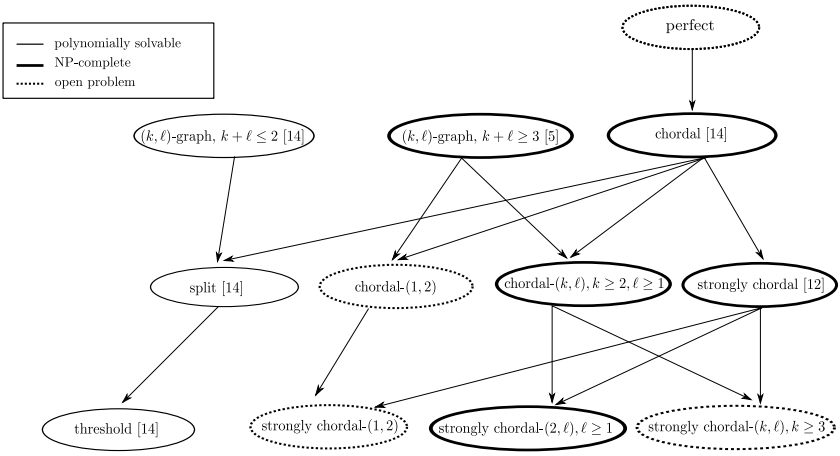


Fig. 4. Complexity of some GRAPH SANDWICH PROBLEMS

Table 1. Complexity results and open problems for $(\Pi_1, (k,\ell), \Pi_2)$-SP, where properties Π_1, Π_2 are in {POLY-COLOR(k), PNMC,$*$}

$(\Pi_1, (k,\ell), \Pi_2) - SP$					
$\Pi_1 \backslash \Pi_2$		PNMC			$*$
		chordal	$\triangle$-free	Δ bounded	
POLY-COLOR(k)	chordal	P	P	P	?
	cograph	P	P	P	?
	comparability	P	P	P	?
$*$		?	?	?	P, if $k+\ell \leq 2$; NP-c, otherwise [4]

References

1. Brandstadt, A.: Partitions of graphs into one or two independent sets and cliques. Discrete Mathematics 152(1-3), 47–54 (1996)
2. Brandstadt, A.: Corrigendum. Discrete Mathematics 186, 295 (2005)
3. Brandstadt, A., Le, V.B., Szymczak, T.: The complexity of some problems related to graph 3-colorability. Discrete Applied Mathematics 89(1-3), 59–73 (1998)
4. Dantas, S., de Figueiredo, C.M., Faria, L.: On decision and optimization (k,l)-graph sandwich problems. Discrete Applied Mathematics 143, 155–165 (2004)
5. Dantas, S., Klein, S., de Mello, C.P., Morgana, A.: The graph sandwich problem for P4-sparse graphs. Discrete Mathematics 309(11), 3664–3673 (2009)
6. Dantas, S., Teixeira, R.B., Figueiredo, C.M.H.: The polynomial dichotomy for three nonempty part sandwich problems. Discrete Applied Mathematics, 1286–1304 (2010)

7. Dourado, M., Petito, P., Teixeira, R.B., Figueiredo, C.M.H.: Helly property, clique graphs, complementary graph classes, and sandwich problems. Journal Brazilian Computer Society 14, 45–52 (2008)
8. Farber, M.: Applications of Linear Programming Duality to Problems Involving Independence and Domination, Ph.d. thesis, Simon Fraser University, Canada (1981)
9. Feder, T., Hell, P., Klein, S., Motwani, R.: List partitions. SIAM Journal Discrete Mathematics 16, 449–478 (2003)
10. Feder, T., Hell, P., Klein, S., Nogueira, L.T., Protti, F.: List matrix partitions of chordal graphs. Theoretical Computer Science 349, 52–66 (2005)
11. Figueiredo, C.M.F., Faria, L., Klein, S., Sritharan, R.: On the complexity ofthe sandwich problems for strongly chordal graphs and chordal bipartite graphs. Theoretical Computer Science 381, 57–67 (2007)
12. Golumbic, M.C., Kaplan, H., Shamir, R.: Graph sandwich problems. Journal of Algorithms 19(3), 449–473 (1995)
13. Hell, P., Klein, S., Nogueira, L.T., Protti, F.: Partitioning chordal graphs into independent sets and cliques. Discrete Applied Mathematics 141, 185–194 (2004)
14. Hell, P., Klein, S., Nogueira, L.T., Protti, F.: Packing r-cliques in weighted chordal graphs. Annals of OR 138, 179–187 (2005)
15. Lueker, G., Rose, D., Tarjan, R.E.: Algorithmic aspects of vertex elimination on graphs. SIAM Journal on Computing 5(2), 266–283 (1976)
16. Nogueira, L.T.: Grafos split e grafos split generalizados, M.sc. thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil (1999)
17. Sritharan, R.: Chordal bipartite completion of colored graphs. Discrete Mathematics 308, 2581–2588 (2008)
18. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM Journal on Computing 13, 566–579 (1984)
19. Tsukiyama, S., Ide, M., Arujoshi, H., Ozaki, H.: A new algorithm for generating all the maximal independent sets. SIAM J. Comput. 6, 505–517 (1977)

Fixed-Parameter Tractability of Workflow Satisfiability in the Presence of Seniority Constraints

J. Crampton[1], R. Crowston[1], G. Gutin[1], M. Jones[1], and M.S. Ramanujan[2]

[1] Royal Holloway University of London, UK
{Jason.Crampton,R.Crowston,G.Gutin,M.E.L.Jones}@rhul.ac.uk
[2] The Institute of Mathematical Sciences, Chennai, India
msramanujan@imsc.res.in

Abstract. The workflow satisfiability problem is concerned with determining whether it is possible to find an allocation of authorized users to the steps in a workflow in such a way that all constraints are satisfied. The problem is NP-hard in general, but is known to be fixed-parameter tractable for certain classes of constraints. In this paper, we provide the first results that establish fixed-parameter tractability of the satisfiability problem when the constraints are asymmetric. We also establish a lower bound for the hardness of the workflow satisfiability problem.

1 Introduction

A business process is a collection of interrelated steps that are performed by users in some (partially) predetermined sequence in order to achieve some objective. In many situations, we wish to restrict the users that can perform certain steps. First, we may wish to specify which users are authorized to perform particular steps. In addition, we may wish, either because of the particular requirements of the business logic or because of statutory requirements, to prevent certain combinations of users from performing particular combinations of steps [5].

We model a *workflow schema* as a tuple (S, U, A, C), where S is the set of steps in the workflow, U is the set of users, $A \subseteq S \times U$ is the *authorization relation*, and C is a set of constraints. Each constraint has the form (ρ, s, s'), where $\rho \subseteq U \times U$ and $s, s' \in S$. A *plan* is a function $\pi : S \to U$. We say π is *valid* (with respect to the schema) if and only if $(s, \pi(s)) \in A$ for all s, and for every constraint $(\rho, s, s') \in C$, $(\pi(s), \pi(s')) \in \rho$. If a workflow schema has a valid plan then we say it is *satisfiable*. Henceforth, we write $L(s) \subseteq U$ to denote the set $\{u \in U : (s, u) \in A\}$; that is, $L(s)$ represents the set of users authorized to perform step s.

For a set, $\{\rho_1, \ldots, \rho_t\}$, of binary relations on U, an instance $\mathcal{I}$ of the *workflow satisfiability problem* WSP$(\rho_1, \ldots, \rho_t)$ is given by a list $L(s)$ for each $s \in S$ and a set C of constraints of the form (ρ, s, s'), where $s, s' \in S$ and $\rho \in \{\rho_1, \ldots, \rho_t\}$. If $\mathcal{I}$ has a valid plan, it is called a YES-instance; otherwise, it is a NO-instance.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 198–209, 2013.

Let $(U, <)$ be the partially ordered set of users. The order relation models the relative seniority of users, $u < v$ indicating that v is senior to u.[1] We write $=$ to denote the binary relation $\{(u, u) : u \in U\}$ and $\neq$ to denote the relation $(U \times U) \setminus \{(u, u) : u \in U\}$.

A brute-force approach to answering WSP gives rise to an algorithm that has running time $O(cn^k)$, where c is the number of constraints[2], n is the number of users and k is the number of steps. Moreover, it is known that determining the satisfiability of a workflow specification is NP-hard in general [14]. It has been shown that WSP$(=, \neq)$ is fixed-parameter tractable (FPT) [6,7,14]. It is also known that WSP$(\neq)$ is equivalent to a problem studied by Fellows *et al.* [8].

In the full version of this paper [4], we establish that the Hasse diagram of $(U, <)$ has treewidth 3 for the Royal Holloway University of London management hierarchy. This seems to be not exceptional in the following sense: it is unlikely that a member of staff will have many line managers (quite often there is only one or two line managers). Thus, it does not seem unreasonable to expect the Hasse diagram of the corresponding partial order to have bounded treewidth and for the treewidth to be rather small. Moreover, our Royal Holloway example indicates that construction of (near-)optimal tree decompositions for such hierarchies may be not hard (see [4]).

In this paper we show that if the Hasse diagram of $(U, <)$ has bounded treewidth then WSP$(=, \neq, <)$ is FPT. This result is based on the tree decomposition of the graph of the seniority relation and the application of dynamic programming to a particular form of tree decomposition. Moreover, we show that it is impossible to obtain an algorithm for the general case of WSP with running time $f(k)n^{o(\frac{k}{\log k})}$ unless the Exponential Time Hypothesis (ETH) [10] fails, where f is an arbitrary function.

In Section 2, we describe tree decompositions, define treewidth and establish some elementary, preparatory results. Section 3 establishes fixed-parameter tractability of the above-mentioned "treewidth bounded" case of the problem and the following section establishes a lower bound for the complexity of the general problem (assuming ETH holds). The lower bound is proved using a result of Marx [12]. We conclude the paper with a summary of our contributions.

2 Preliminaries

Let $(U, <)$ be a partially ordered set. The directed, acyclic graph H with vertex set U such that $x < y$ if and only if there is an arc xy in H is called the *full graph* of $(U, <)$. We write $x \lessdot y$ if $x < y$ and there does not exist $z \in U$ such that $x < z < y$. The directed, acyclic graph D with vertex set U such that $x \lessdot y$ if and only if there is an arc xy in D is called the *Hasse diagram* of $(U, <)$; we may also refer to D as the *reduced graph*. A *mixed graph* consists of a set of vertices together with a set of undirected edges and a set of directed arcs.

[1] Results for constraints of the form $(<, s, s')$ are of practical interest because there are many business rules that require such constraints: the rule that an expenses claim must be authorized by someone senior to the claimant is but one obvious example.

[2] Here and in the rest of the paper, we assume a constraint can be checked in constant time.

2.1 Constraint Graphs

We may represent the set of constraints in an instance $\mathcal{I}$ of WSP$(=, \neq, <)$ with a mixed graph as follows. First, we eliminate constraints of the form $(=, s', s'')$. Specifically, we construct a graph P with vertices S in which $s', s'' \in S$ are adjacent if $\mathcal{I}$ has a constraint $(=, s', s'')$. Observe that the same user must necessarily be assigned to all steps in a connected component Q of P. Thus, if there is a pair $s', s'' \in Q$ such that $\mathcal{I}$ has a constraint $(\neq, s', s'')$ or $(<, s', s'')$, then clearly $\mathcal{I}$ is a NO-instance; thus we may assume that there is no such pair for any connected component of P. For each connected component Q of P, replace all steps of Q in S by a "superstep" q. A user u is authorized to perform q if u is authorized to perform all steps of Q. That is, $L(q) = \bigcap_{s \in Q} L(s)$.

The above procedure eliminates all constraints of the type $(=, s', s'')$ for the reduced set S of steps. All constraints of the types $(\neq, s', s'')$ and $(<, s', s'')$ remain, but steps s' and s'' are replaced by the corresponding supersteps. For simplicity of notation, we will denote the new instance of the problem also by $\mathcal{I}$.

Now we construct a mixed graph with vertex set S. For each constraint of the type $(\neq, s', s'')$, add an *edge* between s' and s''. For each constraint of the type $(<, s', s'')$, add an *arc* from s' and s''. We will refer to the resulting graph as the *constraint graph* (of $\mathcal{I}$). We will say an edge or arc in a constraint graph is *satisfied* by a plan π if π satisfies the corresponding constraint.

It is worth noting that WSP$(\neq)$ is rather closely related to LIST COLORING, where users are "colors" and lists $L(s)$ impose restrictions on the "colors" that can be assigned to steps s. In fact, WSP$(<, \neq)$ is an even more complex problem because it imposes a structure on the set of colors that are available, meaning that the selection of a color for s may preclude the use of many other colors for steps connected to s by an arc.

2.2 Tree Decompositions and Treewidth

Tree decompositions provide a means of representing an undirected or directed graph using a tree. The treewidth of the Hasse diagram of $(U, <)$ determines, in part, the complexity of WSP$(\neq, <)$.

Definition 1. *A* tree decomposition *of an undirected (directed) graph $G = (V, E)$ is a pair $(\mathcal{T}, \mathcal{X})$, where $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ is a tree and $\mathcal{X} = \{\mathcal{B}_i : i \in V_\mathcal{T}\}$ is a collection of subsets of V called* bags*, such that:*

1. $\bigcup_{i \in V_\mathcal{T}} \mathcal{B}_i = V$;
2. *For every edge (arc) $xy \in E$, there exists $i \in V_\mathcal{T}$ such that $\{x, y\} \subseteq \mathcal{B}_i$;*
3. *For every $x \in V$, the set $\{i : x \in \mathcal{B}_i\}$ induces a connected subtree of $\mathcal{T}$.*

The width *of $(\mathcal{T}, \mathcal{X})$ is $\max_{i \in V_\mathcal{T}} |\mathcal{B}_i| - 1$. The* treewidth *of a graph G is the minimum width of all tree decompositions of G.*

To distinguish between vertices of G and $\mathcal{T}$, we call vertices of $\mathcal{T}$ *nodes*. We will often speak of a bag $\mathcal{B}$ interchangeably with the node to which it corresponds in $\mathcal{T}$. Thus, for example, we might say two bags are *neighbors* if they correspond to nodes in $\mathcal{T}$ which are neighbors. We define the *descendants* of a bag $\mathcal{B}$ as follows: every child of $\mathcal{B}$ is a

descendant of $\mathcal{B}$, and every child of a descendant of $\mathcal{B}$ is a descendant of $\mathcal{B}$. We will write $\mathcal{B} = \mathcal{B}'$ if $\mathcal{B}$ and $\mathcal{B}'$ contain the same vertices, while still treating them as different bags.

The problem of determining whether the treewidth of a graph G is at most r (when r is part of input) is NP-complete [1]. Bodlaender [2] obtained an algorithm with running time $O(f(r)n)$ for deciding whether the treewidth of a graph G is at most r, where n is the number of vertices in G and f is a function depending only on r. This algorithm constructs the corresponding tree decomposition with $O(n)$ nodes, if the answer is YES. Unfortunately, f grows too fast to be of practical interest. However, there are several polynomial-time approximation algorithms and heuristics for computing the treewidth of a graph and the corresponding tree decomposition, see, e.g., [3].

We now describe a special type of tree decomposition that is widely used to construct dynamic programming algorithms for solving problems on graphs. A *nice tree decomposition* identifies one node as the root of $\mathcal{T}$, and each node $i \in V_{\mathcal{T}}$ is of one of the following four types:

1. a *join node* $\mathcal{B}$ has two children $\mathcal{B}'$ and $\mathcal{B}''$, with $\mathcal{B} = \mathcal{B}' = \mathcal{B}''$;
2. a *forget node* $\mathcal{B}$ has one child $\mathcal{B}'$, and there exists $u \in \mathcal{B}'$ such that $\mathcal{B} = \mathcal{B}' \setminus \{u\}$;
3. an *introduce node* $\mathcal{B}$ has one child $\mathcal{B}'$, and there exists $u \notin \mathcal{B}'$ such that $\mathcal{B} = \mathcal{B}' \cup \{u\}$;
4. a *leaf node* $\mathcal{B}$ is a leaf of $\mathcal{T}$.

The following useful lemma, concerning the construction of a nice tree decomposition from a given tree decomposition, was proved by Kloks [11, Lemma 13.1.3].

Lemma 1. *Given a tree decomposition with $O(n)$ nodes of a graph G with n vertices, we can construct, in time $O(n)$, a nice tree decomposition of G of the same width and with at most $4n$ nodes.*

Lemma 2. *Let D be a (directed) graph, $(\mathcal{T}, \mathcal{X})$ a tree decomposition of D, and let Y be a set of vertices in D such that $D[Y]$ is connected. Then the set of bags containing vertices in Y induces a connected subtree in $\mathcal{T}$.*

Proof. The proof is by induction on $|Y|$. The base case, $|Y| = 1$, follows from Definition 1. Let $y \in Y$ such that $D[Y \setminus \{y\}]$ is connected and suppose that the set of bags containing vertices in $Y \setminus \{y\}$ induces a connected subtree $\mathcal{T}'$ of $\mathcal{T}$. Let $z \in Y$ such that yz is an edge of D. By Definition 1, y and z belong to the same bag $\mathcal{B}$ and observe that $\mathcal{B}$ is in $\mathcal{T}'$. Thus, the subtree of $\mathcal{T}$ induced by the bags containing y and $\mathcal{T}'$ intersect and so the set of bags containing vertices in Y induces a connected subtree in $\mathcal{T}$. □

For a digraph D and a pair of verices $u \neq v$ in D, we say that a set $X \subseteq V(D) \setminus \{u, v\}$ *separates u from v* if $D - X$ has no directed path from u to v.

Lemma 3. *Let D be the reduced graph for $(U, <)$. Let u, v be users and $\mathcal{B}$ a set of users such that $u \neq v$ and $u, v \notin \mathcal{B}$, and $\mathcal{B}$ separates u from v in D. Then $u < v$ if and only if there exists $w \in \mathcal{B}$ such that $u < w$ and $w < v$.*

Proof. By transitivity, if $u < w < v$ then $u < v$. For the other direction, suppose $u < v$. Then by the definition of D there must exist a directed path from u to v in D. Since $\mathcal{B}$ separates u and v, this path must contain a user w in $\mathcal{B}$. Therefore $u < w$ and $w < v$. □

3 FPT Algorithm for Bounded Treewidth

In this section, we consider the special case of the problem when the reduced graph D of $(U, <)$ is of treewidth bounded by a constant r. Note that D may have much smaller treewidth than the full graph H. For example, when $<$ is a linear order on U, then H is a tournament with treewidth $|U| - 1$, but D is a directed path with treewidth 1.

Theorem 1. *Let $\mathcal{I}$ be an instance of* WSP$(=, \neq, <)$ *and let D be the reduced graph of $(U, <)$. Given a tree decomposition of D of treewidth r and with $O(n)$ nodes, we can solve $\mathcal{I}$ in time $O(nk4^k(r + 2 + 3^{r+1})^k)$, where k is the number of steps and n is the number of users.*

By Lemma 1, assume we have a nice tree decomposition $(\mathcal{T}, \mathcal{X})$ of D of width r and with at most $4n$ nodes. Before proving the above result, we provide an informal insight into our approach. We use dynamic programming techniques to compute solutions to restricted instances of the original problem instance, and for each of these restricted instances, we construct possible intermediate solutions for each bag in the nice tree decomposition. Working from the leaves of the decomposition back to the root, we extend intermediate solutions for child nodes to an intermediate solution for the parent node. The existence of an intermediate solution for the root node, implies the existence of a solution for the original problem instance (Lemma 4). Then, in Lemma 5, we establish the complexity of computing an intermediate solution, thereby completing the proof of Theorem 1. Roughly speaking, for every subset T of the set S of steps, each bag $\mathcal{B}$ in the tree decomposition of D, and each step x in T, we keep track of which user in $\mathcal{B}$, if any, x is to be assigned to, and otherwise what relation the user assigned to x should have to the users in $\mathcal{B}$. Before proceeding further, we introduce some definitions and notation.

Let us say that $u > v$ if $v < u$, and $u \sim v$ if neither $u < v$ nor $v < u$. Define the *relation* of v to u, a function $\phi(v, u)$ from the set of all pairs of users to the set of three symbols $[<], [>], [\sim]$, as follows:

$$\phi(v,u) = \begin{cases} [<] \text{ if } v < u \\ [>] \text{ if } v > u \\ [\sim] \text{ if } v \sim u. \end{cases}$$

For each bag $\mathcal{B} = \{u_1, u_2, \ldots, u_p\}$ in $\mathcal{X}$, and each user $v \notin \mathcal{B}$, define the *relation* of v to $\mathcal{B}$, $\mathcal{R}(v, \mathcal{B})$ to be the ordered tuple $(\phi(v, u_1), \ldots, \phi(v, u_p))$.

Definition 2. *Given a workflow instance $\mathcal{I}$ with constraint graph $G = (S, E)$, a bag $\mathcal{B}$ in the nice tree decomposition of $(U, <)$, a set of steps T and a function $R : T \to \mathcal{B} \cup \{[<], [>], [\sim]\}^{|\mathcal{B}|}$, we say $\pi : T \to U$ is a* $(\mathcal{B}, T, R)$-plan *if the following conditions are satisfied:*

1. *$\pi(x) \in L(x)$ for each $x \in T$;*
2. *if there is an edge between x and y in $G[T]$, then $\pi(x) \neq \pi(y)$;*
3. *if there is an arc from x to y in $G[T]$, then $\pi(x) < \pi(y)$;*
4. *for each step x, $\pi(x)$ is either a user in $\mathcal{B}$ or a user in a descendant of $\mathcal{B}$;*

5. *for any* $x \in T$, $u \in \mathcal{B}$, $\pi(x) = u$ *if and only if* $R(x) = u$;
6. *if* $R(x) \notin \mathcal{B}$, *then* $\mathcal{R}(\pi(x), \mathcal{B}) = R(x)$.

R provides a partial allocation of users in $\mathcal{B}$ to steps in T; where no user is allocated, R identifies the relationships that must hold between the user that is subsequently allocated to the task and those users in $\mathcal{B}$. The existence of a $(\mathcal{B}, T, R)$-plan means that we can extend R to a full plan π by traversing the nice tree decomposition.

We may now define the function that is central to our dynamic programming approach. For every bag $\mathcal{B}$ in the tree decomposition of D, every subset T of S, and every possible function $R : T \to \mathcal{B} \cup \{[<], [>], [\sim]\}^{|\mathcal{B}|}$, define $F(\mathcal{B}, T, R) =$ TRUE if there exists a $(\mathcal{B}, T, R)$-plan and FALSE otherwise.

Lemma 4. *Let* $\mathcal{B}_0$ *be the root node in the nice tree decomposition of* D. *Then* $\mathcal{I}$ *is a* YES*-instance if and only if there exists a function* $R : S \to \mathcal{B}_0 \cup \{[<], [>], [\sim]\}^{|\mathcal{B}_0|}$ *such that* $F(\mathcal{B}_0, S, R) =$ TRUE.

Proof. By the first three conditions on $F(\mathcal{B}_0, S, R)$ being TRUE and the definition of the constraint graph G, it is clear that if $F(\mathcal{B}_0, S, R) =$ TRUE for some R then we have a YES-instance. So now suppose $\mathcal{I}$ is a YES-instance, and let $\pi : S \to U$ be a valid plan. Then for each $x \in S$, let $R(x) = \pi(x)$ if $\pi(x) \in \mathcal{B}_0$, and otherwise, let $R(x) = \mathcal{R}(\pi(x), \mathcal{B})$. Then observe that all the conditions on $F(\mathcal{B}_0, S, R)$ being TRUE are satisfied and therefore $F(\mathcal{B}_0, V, R) =$ TRUE. □

Lemma 5. *We can compute* $F(\mathcal{B}, T, R)$ *for every bag* $\mathcal{B}$ *in* $\mathcal{X}$, *every* $T \subseteq S$, *and every* $R : T \to \mathcal{B} \cup \{[<], [>], [\sim]\}^{|\mathcal{B}|}$ *in time* $O(nk4^k(r + 2 + 3^{r+1})^k)$.

Proof. We will start by constructing, in advance, a matrix $\mathcal{L} = [L_{s,u}]_{s \in S, u \in U}$ such that $L_{s,u} = 1$ if $u \in L(s)$ and $L_{s,u} = 0$, otherwise. This will take time $O(kn)$. Let $\mathcal{B}$ be in $\mathcal{X}$, T a subset of S, and R a function from T to $\mathcal{B} \cup \{[<], [>], [\sim]\}^{|\mathcal{B}|}$. Recall that every bag $\mathcal{B}$ is either a leaf node, a join node, a forget node or an introduce node. We will consider the four possibilities separately.

$\mathcal{B}$ Is a Leaf Node. Since $\mathcal{B}$ has no descendants, $F(\mathcal{B}, T, R) =$ FALSE unless $R(x) \in \mathcal{B}$ for every $x \in T$. So now we may assume $R(x) \in \mathcal{B}$ for all x. But then the only possibility for a $(\mathcal{B}, T, R)$-plan is one in which $\pi(x) = R(x)$ for all x. Therefore we may check, in time $O(k^2)$, whether this plan satisfies the (first three) conditions on $F(\mathcal{B}, T, R)$ being TRUE. (Use matrix $\mathcal{L}$ to check that $\pi(x) \in L(x)$ for all $x \in T$.) If it does, $F(\mathcal{B}, T, R) =$ TRUE, otherwise $F(\mathcal{B}, T, R) =$ FALSE.

For the remaining cases, we may assume that $F(\mathcal{B}', T, R)$ has been calculated for every child of $\mathcal{B}'$ of $\mathcal{B}$ and every possible T, R.

$\mathcal{B}$ Is a Forget Node. Let $\mathcal{B}' = \{u_1, u_2, \ldots, u_p\}$ be the child node of $\mathcal{B}$ and assume without loss of generality that $\mathcal{B} = \{u_1, u_2, \ldots, u_{p-1}\}$. For $i \in [p-1]$, let X_i be the set of steps in T with $R(x) = u_i$.

Suppose that π is a $(\mathcal{B}, T, R)$-plan. Then let $R' : T \to \mathcal{B}' \cup \{[<], [>], [\sim]\}^{|\mathcal{B}'|}$ be the function such that $R'(x) = \pi(x)$ if $\pi(x) \in \mathcal{B}'$, and $R'(x) = \mathcal{R}(\pi(x), \mathcal{B}')$ if $\pi(x) \notin \mathcal{B}'$. It is clear that $F(\mathcal{B}', T, R') =$ TRUE. Now we show some properties of R.

Firstly, since π is a $(\mathcal{B}, T, R)$-plan, it must be the case that $\pi(x) = R(x)$ if $R(x) \in \mathcal{B}$ and therefore $R'(x) = R(x)$ if $R(x) \in \mathcal{B}$. Secondly, since π is a $(\mathcal{B}, T, R)$-plan and $u_p \notin \mathcal{B}$, it must be the case that $\pi(x) = u_p$ only if $R(x) = \mathcal{R}(u_p, \mathcal{B})$. Therefore $R'(x) = u_p$ only if $R(x) = \mathcal{R}(u_p, \mathcal{B})$. Finally, for $x \in T$ with $R'(x) \notin \mathcal{B}'$, let $R(x) = (x_{u_1}, x_{u_2}, \ldots, x_{u_{p-1}})$ and let $R'(x) = (x'_{u_1}, x'_{u_2}, \ldots, x'_{u_p})$. Since π is a $(\mathcal{B}, T, R)$-plan and a $(\mathcal{B}', T, R')$-plan, we must have that $x_{u_i} = \phi(\pi(x), u_i) = x'_{u_i}$ for all $i \in [p-1]$. That is $R(x)$ and $R'(x)$ are the same except that $R'(x)$ has the extra co-ordinate x'_{u_p}. It follows that to obtain R' from R, we merely need to guess which x with $R(x) = \mathcal{R}(u_p, \mathcal{B})$ are assigned to u_p by R', and for all other x, what the value of x_{u_p} should be.

Therefore, in order to calculate $F(\mathcal{B}, T, R)$, we may do the following: Try every possible way of partitioning $T \setminus (X_1 \cup X_2 \cup \cdots \cup X_{p-1})$ into four sets $X_p, X_<, X_>, X_\sim$, subject to the constraint that $x \in X_p$ only if $R(x) = \mathcal{R}(u_p, \mathcal{B})$. For each such partition, construct a function $R' : T \to \mathcal{B}' \cup \{[<], [>], [\sim]\}^{|\mathcal{B}'|}$ such that

1. $R'(x) = R(x)$ if $R(x) \in \mathcal{B}$.
2. $R'(x) = u_p$ if $x \in X_p$.
3. For all other x, let $R(x) = (x_{u_1}, x_{u_2}, \ldots, x_{u_{p-1}})$. Then $R'(x) = (x'_{u_1}, x'_{u_2}, \ldots, x'_{u_p})$, where $x'_{u_i} = x_{u_i}$ for all $i \in [p-1]$, and $x'_{u_p} = [<]$ if $x \in X_<$, $x'_{u_p} = [>]$ if $x \in X_>$, and $x'_{u_p} = [\sim]$ if $x \in X_\sim$.

and check the value of $F(\mathcal{B}', T, R')$.

By the above argument, if $F(\mathcal{B}, T, R) = \textsc{True}$ then it must be the case that $F(\mathcal{B}', T, R') = \textsc{True}$ for one of the R' constructed in this way. Therefore if $F(\mathcal{B}', T, R') = \textsc{False}$ for all such R', we know that $F(\mathcal{B}, T, R) = \textsc{False}$. Otherwise, if $F(\mathcal{B}', T, R') = \textsc{True}$ for some R', let π be a $(\mathcal{B}', T, R')$-plan, and observe that by construction of R' and X_p, π is a $(\mathcal{B}, T, R)$-plan as well. Therefore $F(\mathcal{B}, T, R) = \textsc{True}$.

Finally, observe that there are at most 4^k possible values of R' to check and each R' can be constructed in time $O(k)$, and therefore we can calculate $F(\mathcal{B}, T, R)$ in time $O(k4^k)$.

$\mathcal{B}$ Is an Introduce Node. Let $\mathcal{B} = \{u_1, u_2, \ldots, u_p\}$, let $\mathcal{B}'$ be the child node of $\mathcal{B}$ and assume without loss of generality that $\mathcal{B}' = \{u_1, u_2, \ldots, u_{p-1}\}$. Let $X_p \subseteq T$ be the set of all $x \in T$ with $R(x) = u_p$, and let $T' = T \setminus X_p$. Define a function $R' : T' \to \mathcal{B}' \cup \{[<], [>], [\sim]\}^{|\mathcal{B}'|}$ as follows:

1. $R'(x) = R(x)$ if $R(x) \in \mathcal{B}'$.
2. For all other x, let $R(x) = (x_{u_1}, x_{u_2}, \ldots, x_{u_p})$. Then set $R'(x) = (x'_{u_1}, x'_{u_2}, \ldots, x'_{u_{p-1}})$, where $x'_{u_i} = x_{u_i}$ for all $i \in [p-1]$.

We will now give eight conditions which are necessary for $F(\mathcal{B}, T, R) = \textsc{True}$. We will then show that these conditions collectively are sufficient for $F(\mathcal{B}, T, R) = \textsc{True}$. Since each of these conditions can be checked in time $O(k^2)$, we will have that $F(\mathcal{B}, T, R)$ can be calculated in time $O(k^2)$.

Condition 1*: $u_p \in L(x)$ for each $x \in X_p$.* This condition is clearly necessary, as for every $(\mathcal{B}, T, R)$-plan π we have $\pi(x) = u_p$.

Condition 2*: X_p is an independent set in G.* Since in any $(\mathcal{B}, T, R)$-plan, all steps in X_p must be assigned the same user, any arc or edge between steps in X_p will not be satisfied.

Condition 3*: If there exists $x \in X_p$, $y \notin X_p$ with an arc from y to x in G, then either $R(y) = u_i$ for some $u_i \in \mathcal{B}'$ with $u_i < u_p$, or $R(y) = (y_{u_1}, \ldots, y_{u_p})$ with $y_{u_p} = [<]$.* For if not, then any $(\mathcal{B}, T, R)$-plan will assign y to a user v such that $v > u_p$ or $v \sim u_p$, and the arc yx will not be satisfied.

Condition 4*: If there exists $x \in X_p$, $y \notin X_p$ with an arc from x to y in G, then either $R(y) = u_i$ for some $u_i \in \mathcal{B}'$ with $u_i > u_p$, or $R(y) = (y_{u_1}, \ldots, y_{u_p})$ with $u_p = [>]$.* The proof is similar to the proof of Condition 3.

Condition 5*: If there exists $y \notin X_p$ such that $R(y) = (y_{u_1}, \ldots, y_{u_p})$ with $y_{u_p} = [<]$, then there must exist $u_i \in \mathcal{B}'$ with $y_{u_i} = [<]$ and $u_i < u_p$.* For suppose there is a $(\mathcal{B}, T, R)$-plan π, and let $v = \pi(y)$. Note that v must be in a descendant of $\mathcal{B}$ but not in $\mathcal{B}'$. Therefore $\mathcal{B}'$ separates v from u_p in D, for any v in a descendant of $\mathcal{B}$. (This follows from Lemma 2 where Y is the vertices of a path between v and u_p). Then by Lemma 3, as $v < u_p$ there exists $u_i \in \mathcal{B}'$ with $v < u_i < u_p$. Therefore $y_{u_i} = [<]$.

Condition 6*: If there exists $y \notin X_p$ such that $R(y) = (y_{u_1}, \ldots, y_{u_p})$ with $y_{u_p} = [>]$, then there must exist $u_i \in \mathcal{B}'$ with $y_{u_i} = [>]$ and $u_i > u_p$.* The proof is similar to the proof of Condition 5.

Condition 7*: If there exists $y \notin X_p$ such that $R(y) = (y_{u_1}, \ldots, y_{u_p})$ with $y_{u_p} = [\sim]$, then there is no $u_i \in \mathcal{B}$ such that $y_{u_i} = [<]$ and $u_i < u_p$, or $y_{u_i} = [>]$ and $u_i > u_p$.* For suppose there is a $(\mathcal{B}, T, R)$-plan π, and let $v = \pi(y)$. Suppose for a contradiction that there exists $u_i \in \mathcal{B}$ such that $y_{u_i} = [>]$ and $u_i > u_p$. (The case $y_{u_i} = [<]$ and $u_i < u_p$ is handled similarly). Then $v > u_i$ and so by transitivity, $v > u_p$. But this is a contradiction as $y_{u_p} = [\sim]$.

Condition 8*: $F(\mathcal{B}', T', R') =$* TRUE. For suppose π is a $(\mathcal{B}, T, R)$-plan. Then observe that by construction of R', π restricted to T' is a $(\mathcal{B}', T', R')$-plan.

It now remains to show that if Conditions 1-8 hold then $(\mathcal{B}, T, R) =$ TRUE. Let π' be a $(\mathcal{B}', T', R')$-plan whose existence is guaranteed by Condition 8, and let π be the extension of π' to T in which $\pi(x) = u_p$ for all $x \in X_p = T \setminus T'$. Since π' is a $(\mathcal{B}', T', R')$-plan, $\pi(x) \in L(x)$ for all $x \in T'$, and by Condition 1, $\pi(x) \in L(x)$ for all $x \in X_p$. For every x with $R(x) \in \mathcal{B}$, we have that $\pi(x) = R(x)$ by the fact that π' is a $(\mathcal{B}', T', R')$-plan and $R(x) = u_p$ for all $x \in X_p$.

Now consider x with $R(x) \notin \mathcal{B}$. Then let $R(x) = (x_{u_1}, x_{u_2}, \ldots, x_{u_p})$. By construction of R' and the fact that π' is a $(\mathcal{B}', T', R')$-plan, $\phi(\pi(x), u_i) = x_{u_i}$ for $i \in [p-1]$. Suppose $x_{u_p} = [<]$. Then by Condition 5, there exists $u_i \in \mathcal{B}'$ with $x_{u_i} = [<]$ and $u_i < u_p$. Therefore $\pi(x) < u_i$ and so $\pi(x) < u_p$. Therefore $\phi(\pi(x), u_i) = [<]$. Similarly, using Condition 6, if $x_{u_p} = [>]$ then $\phi(\pi(x), \mathcal{B}) = [>]$. If $\phi(\pi(x), u_i) = [\sim]$ then by Condition 7 there is no $u_i \in \mathcal{B}'$ with $\pi(x) > u_i > u_p$ or $\pi(x) < u_i < u_p$. Then by Lemma 3, $\pi(x) \sim u_p$ and so $\phi(\pi(x), u_p) = [\sim]$. In each case we have that $\phi(\pi(x), u_p) = x_{u_p}$ and so $\mathcal{R}(\pi(x), \mathcal{B}) = R(x)$.

It is clear that for each step x, $\pi(x)$ is either in $\mathcal{B}$ or in a descendant of $\mathcal{B}$. It remains to show that the arcs and edges in $G[T]$ are satisfied by π.

As π' is a $(\mathcal{B}', T', R')$-plan, every arc and edge in $G[T']$ is satisfied by π. By Condition 2 there are no edges and arcs within $G[X_p]$. It remains to show that the arcs and

edges between X_p and T' are satisfied by π. Consider an edge between $x \in X_p$ and $y \in T'$. Since $\pi(x) = u_p$, and $\pi(y) \neq u_p$ (since u_p does not appear in $\mathcal{B}'$ or any descendant of $\mathcal{B}'$ by definition of a tree decomposition), this edge is satisfied. Now suppose there is an arc from $y \in T'$ to $x \in X_p$. By Condition 3, either $\pi(y) = R(y) = u_i$ with $u_i < u_p$, or $y_{u_p} = [<]$, in which case $\pi(y) < u_p$ (as we have shown $\phi(\pi(y, u_p) = y_{u_p})$. In either case $\pi(y) < \pi(x)$ and so the arc is satisfied. Similarly, if there is an arc from $x \in X_p$ to $y \in S'$, then by Condition 4 $\pi(y) > \pi(x)$ and the arc is satisfied.

Thus π satisfies all the conditions of a $(\mathcal{B}, T, R)$-plan and so $F(\mathcal{B}, T, R) = \textsc{True}$.

$\mathcal{B}$ Is a Join Node. Let $\mathcal{B}', \mathcal{B}''$ be the two child nodes of $\mathcal{B}$, and recall that $\mathcal{B}'$ and $\mathcal{B}''$ contain the same users as $\mathcal{B}$. Let X be the set of all $x \in T$ with $R(x) \in \mathcal{B}$.

Let π be a $(\mathcal{B}, T, R)$-plan. Then let X' be the set of all $x \in T \setminus X$ such that $\pi(x) = v$ for some v in a descendant of $\mathcal{B}'$, and let X'' be the set of all $x \in T \setminus X$ such that $\pi(x) = v$ for some v in a descendant of $\mathcal{B}''$. (Observe that X, X', X'' is a partition of T.) Let $T' = X \cup X'$ and let R' be the function R restricted to T'. Similarly let $T'' = X \cup X''$ and let R'' be the function R restricted to T''. Then observe that $F(\mathcal{B}', T', R') = \textsc{True}$ and $F(\mathcal{B}'', T'', R'') = \textsc{True}$.

Now consider an arc from $x \in X'$ to $y \in X''$. Then $\pi(x) < \pi(y)$. Since $\mathcal{B}$ separates $\pi(x)$ from $\pi(y)$ (by Lemma 2 with Y the set of vertices on a path between $\pi(x)$ and $\pi(y)$), there must exist $u_i \in \mathcal{B}$ such that $\pi(x) < u_i < \pi(y)$. Therefore $x_{u_i} = [<]$ and $y_{u_i} = [>]$. Similarly, if there is an arc from $y \in X''$ to $x \in X'$ then there exists $u_i \in \mathcal{B}$ with $x_{u_i} = [<]$ and $y_{u_i} = [>]$.

We therefore have that if $F(\mathcal{B}, T, R) = \textsc{True}$, then there exists a partition X', X'' of $T \setminus X$ such that $F(\mathcal{B}', T', R') = \textsc{True}$ and $F(\mathcal{B}'', T'', R'') = \textsc{True}$ (where T', T'', R', R'' are as previously defined) and for any arc from $x \in X'$ to $y \in X''$, there exists $u_i \in \mathcal{B}$ with $x_{u_i} = [<]$ and $y_{u_i} = [>]$ (and similarly for arcs from $y \in X''$ to $x \in X'$). We now show that the converse is true.

Suppose these conditions hold, and let π' be a $(\mathcal{B}', T', R')$-plan and π'' a $(\mathcal{B}'', T'', R'')$-plan. Note that for all $x \in X$, $\pi'(x) = R(x) = \pi''(x)$. Let π be the assignment on S made by combining π' and π'', i.e. $\pi(x) = \pi'(x) = \pi''(x)$ for $x \in X$, $\pi(x) = \pi'(x)$ for $x \in X'$, and $\pi(x) = \pi''(x)$ for $x \in X''$.

Observe that by definition of π' and π'', $L_{x,\pi(x)} = 1$ for all $x \in T$, $\pi(x) = R(x)$ if $R(x) \in \mathcal{B}$, and otherwise $\mathcal{R}(\pi(x), \mathcal{B}) = R(x)$. Any edges and arcs in $G[X \cup X']$ are satisfied by π, by definition of π', and any edges and arcs in $G[X \cup X'']$ are satisfied by π, by definition of π''. It remains to consider the edges and arcs between X' and X''. Since the tasks in X' and X'' are assigned to disjoint sets of users (by Lemma 2), any edge between and X' and X'' is satisfied. If there is an arc from $x \in X'$ to $y \in X''$, then by our assumption there exists $u_i \in \mathcal{B}$ with $x_{u_i} = [<]$ and $y_{u_i} = [>]$. Therefore $\pi(x) < u_i < \pi(y)$, and therefore $\pi(x) < \pi(y)$, and so the arc is satisfied. A similar argument applies when there is an arc from $y \in X''$ to $x \in X'$.

Since there are at most $2^{|T|}$ possible ways to partition $T \setminus X$ into X' and X'', we can calculate $F(\mathcal{B}, T, R)$ in $O(2^k)$ time.

The above bounds show that, provided all the values for descendants of $\mathcal{B}$ have been computed, $F(\mathcal{B}, T, R)$ can be calculated in time $O(k4^k)$, for each possible $\mathcal{B}, T$ and R. It remains to count the number of possible values of $\mathcal{B}, T$ and R. There are at most $4n$

values of $\mathcal{B}$. Calculating $F(\mathcal{B}, T, R)$ for every T and R can be viewed as calculating F for every function $R^* : S \to \mathcal{B} \cup \{[<], [>], [\sim]\}^{|\mathcal{B}|} \cup \{0\}$, T being defined as the set of steps not mapped to 0. Finally, for each step x in S there are $r + 2 + 3^{r+1}$ possible values for $R^*(x)$ and therefore $(r + 2 + 3^{r+1})^k$ possible values for R^*. Therefore the total number of possible values of $F(\mathcal{B}, T, R)$ is $O(n(r + 2 + 3^{r+1})^k)$, and so every value $F(\mathcal{B}, T, R)$ can be calculated in time $O(nk4^k(r + 2 + 3^{r+1})^k)$. □

4 Hardness

The main theorem of this section establishes a lower bound for the complexity of the workflow satisfiability problem. In fact, we show that in general, the trivial $O(n^k)$ algorithm is nearly optimal. Our result assumes the Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi, and Zane [10]: that is, we assume that there is no $2^{o(n)}$-time algorithm for n-variable 3-SAT.

The *degree of a vertex* x in a digraph D is the number of arcs entering x and leaving x. For a natural number ℓ, we say that a digraph D is ℓ-degenerate if each induced subgraph of D has a vertex of degree at most ℓ.

Theorem 2. WSP$(=, \neq, <)$ *cannot be solved in time* $f(k)n^{o(\frac{k}{\log k})}$ *unless* ETH *fails, where* f *is an arbitrary function,* k *is the number of steps and* n *is the number of users. This results holds even if the full graph of* $(U, <)$ *is 2-degenerate.*

In order to prove Theorem 2, we first consider the following problem and prove the following lemma. In what follows, for a natural number m, $[m] = \{1, 2, \ldots, m\}$.

SUBTDAG ISOMORPHISM	
Input:	Transitive acyclic digraphs $D = (V_D, A_D)$ and $R = (V_R, A_R)$, a subset $W_R = \{w_1, \ldots, w_{\lvert W_R\rvert}\}$ of V_R, and disjoint subsets $W_{D,1}, \ldots, W_{D,\lvert W_R\rvert}$ of V_D.
Parameter:	$\lvert V_R\rvert$
Question:	Is there an injection $\gamma : V_R \to V_D$ such that $\gamma(w_i) \in W_{D,i}$ for each $i \in [\lvert W_R\rvert]$, and for every $(u, v) \in A_R$, $(\gamma(u), \gamma(v)) \in A_D$?

Lemma 6. SUBTDAG ISOMORPHISM *cannot be solved in time* $f(k)n^{o(\frac{k}{\log k})}$ *where* f *is an arbitrary function,* n *is the number of vertices in* D *and* k *is the number of vertices in* R*, unless* ETH *fails. This result holds even if* D *and* R *are 2-degenerate.*

A proof of this lemma can be found in the full version [4] of this paper. The proof is based on Corollary 6.3 in [12].

Proof of Theorem 2. The proof is by a reduction from the SUBTDAG ISOMORPHISM problem. Let $(D, R, W_R, W_{D,1}, \ldots, W_{D,|W_R|})$ be an instance of SUBTDAG ISOMORPHISM. We construct an instance of WSP$(=, \neq, <)$ as follows. We define the set U of users to be V_D and the set S of steps to be V_R. For every step $w_i \in W_R$, $L(w_i) = W_{D,i}$, and for every step $s \in S \setminus W_R$, $L(s) = U$.

We define the relation $<$ on U as follows. For every $x, y \in U$, $x < y$ if and only if $x \neq y$ and there is a arc from x to y in D. For every arc $(u, v) \in A_R$, we add a constraint $(<, u, v)$ and for every pair u, v of distinct non-adjacent vertices of R, we add a constraint $(\neq, u, v)$. Let the instance of WSP$(=, \neq, <)$ thus constructed be $\mathcal{I}$. We claim that $(D, R, W_R, W_{D,1}, \ldots, W_{D,|W_R|})$is a YES instance of SUBTDAG ISOMORPHISM iff $\mathcal{I}$ is a YES instance of WSP$(=, \neq, <)$.

Suppose that $(D, R, W_R, W_{D,1}, \ldots, W_{D,|W_R|})$ is a YES-instance of SUBTDAG ISOMORPHISM and let γ be a required injection for this instance. We define a plan π as $\pi(v) = \gamma(v)$ for every $v \in S$. It is easy to see that π is an valid plan for $\mathcal{I}$.

Conversely, suppose that $\mathcal{I}$ is a YES-instance of WSP$(=, \neq, <)$ and let π be a valid plan for this instance. We define a function $\gamma : V_R \to V_D$ as follows. For every $u \in V_R$, we set $\gamma(u) = \pi(u)$. It remains to verify that γ is a required injection for the instance $(D, R, W_R, W_{D,1}, \ldots, W_{D,|W_R|})$. We first show that γ is an injection. Suppose this were not the case and let u and v be two distinct vertices such that $\gamma(u) = \gamma(v)$. This implies that $\pi(u) = \pi(v)$. But then this assignment satisfies neither the constraint $(\neq, u, v)$ nor the constraint $(<, u, v)$, which is a contradiction. Hence, we conclude that γ is indeed an injection. Now, consider an arc $(u, v) \in R$. Since π is a valid plan, $\pi(u) < \pi(v)$, which implies that $\gamma(u) < \gamma(v)$, which by definition is possible only if $(\gamma(u), \gamma(v)) \in A_D$. This completes the proof of correctness of the reduction.

It remains to apply Lemma 6 to complete the proof of the theorem. □

It is well-known (see, e.g., [9]) that ETH is stronger than the widely believed complexity hypothesis W[1] $\neq$ FPT. Thus, we have the following:

Corollary 1. WSP$(=, \neq, <)$ *is not FPT unless W[1] = FPT. This results holds even if the full graph of* $(U, <)$ *is 2-degenerate.*

This corollary proves that while the class of treewidth bounded graphs is sufficiently special to imply an FPT algorithm, considering the more general class of graphs of bounded degeneracy does not make the problem any easier.

5 Concluding Remarks

The main contribution of this paper is the development of the first FPT algorithm for WSP$(=, \neq, <)$, where $<$ is a (transitive) relation on the set of users. Unlike WSP$(=, \neq)$ which is FPT in the general case, WSP$(=, \neq, <)$ is not FPT unless W[1]=FPT, which is highly unlikely. In fact, under a stronger hypothesis (ETH) we have shown that we even cannot have an algorithm significantly faster than the trivial brute-force algorithm. Thus, it is natural to identify special cases of WSP$(=, \neq, <)$ that are in FPT and of practical relevance. We have done this by restricting the reduced graph D of $(U, <)$ to lie in the class of graphs of bounded treewidth. We believe that this restriction on treewidth holds for many user hierarchies that arise in practice. On the other hand, we have also shown that the restriction of the reduced (or even full) graph to the class of 2-degenerate graphs does not reduce the complexity of the problem.

Unfortunately, it seems that our FPT algorithm is efficient in practice only for rather small values of the number of steps k and the treewidth r of D (we may view $k + r$ as a

combined parameter). However, it is quite often the case that the first FPT algorithm for a parameterized problem is not efficient except for rather small values of the parameter, but subsequent improvements bring about an FPT algorithm efficient for larger values of the parameter [9,13]. We believe that a more efficient FPT algorithm for $\mathrm{WSP}(=, \neq, <)$ may be possible.

Acknowledgment. This research was partially supported by an EPSRC grant EP/K005162/1 and International Joint grant of the Royal Society.

References

1. Arnborg, S., Corneil, D., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discrete Methods 8(2), 277–284 (1987)
2. Bodlaender, H.: A linear time algorithm for finding tree decompositions of small treewidth. SIAM J. Comput. 25, 1305–1317 (1996)
3. Bodlaender, H., Koster, A.: Treewidth computations I. Upper bounds. Information and Computation 208(3), 259–275 (2010)
4. Crampton, J., Crowston, R., Gutin, G., Jones, M., Ramanujan, M.: Fixed-parameter tractability of workflow satisfiability in the presence of seniority constraints. CoRR abs/1210.3978 (2012)
5. Crampton, J.: A reference monitor for workflow systems with constrained task execution. In: SACMAT 2005, pp. 38–47 (2005)
6. Crampton, J., Gutin, G.: Constraint Expressions and Workflow Satisfiability. In: SACMAT 2013 (to appear, 2013)
7. Crampton, J., Gutin, G., Yeo, A.: On the Parameterized Complexity and Kernelization of the Workflow Satisfiability Problem. ACM Trans. Inform. System & Secur. (to appear), Preliminary version in. ACM Conf. Comput. & Communic. Secur., pp. 857–868 (2012)
8. Fellows, M.R., Friedrich, T., Hermelin, D., Narodytska, N., Rosamond, F.A.: Constraint satisfaction problems: Convexity makes AllDifferent constraints tractable. Theor. Comput. Sci. 472, 81–89 (2013)
9. Flum, J., Grohe, M.: Parameterized Complexity Theory. In: Texts in Theoretical Computer Science. Springer (2006)
10. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. 63(4), 512–530 (2001)
11. Kloks, T. (ed.): Treewidth. Computations and Approximations. LNCS, vol. 842. Springer, Heidelberg (1994)
12. Marx, D.: Can you beat treewidth? Theory of Computing 6(1), 85–112 (2010)
13. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
14. Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. ACM Trans. Inf. Syst. Secur. 13(4), 40 (2010)

Two-Round Discrete Voronoi Game along a Line

Aritra Banik[1], Bhaswar B. Bhattacharya[2], Sandip Das[1], and Sreeja Das[3]

[1] Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India
aritrabanik@gmail.com, sandipdas@isical.ac.in
[2] Department of Statistics, Stanford University, USA
bhaswar@stanford.edu
[3] Jadavpur University, Kolkata, India
sreejadas.jucse@gmail.com

Abstract. The two-round discrete Voronoi game on a line consists of a finite user set U (with $|U| = n$), placed along a line ℓ, and two players Player 1 ($P1$) and Player 2 ($P2$). We assume that the sorted order of users in U along the line ℓ is known, and $P1$ and $P2$ each has two facilities. $P1$ places one facility followed by which $P2$ places another facility and this continues for two rounds. The payoff of $P2$ is defined as the cardinality of the set of points in U which are closer to a facility owned by $P2$ than to every facility owned by $P1$. The payoff of $P1$ is the number of users in U minus the payoff of $P2$. The objective of both the players is to maximize their respective payoffs. In this paper we show that, $P2$ always gets at least $n/2$ users, i.e., $P2$ always wins the game and the bound is tight. We also present efficient algorithms to find the optimal strategies of the players in both the rounds.

1 Introduction

The Voronoi game is a variant of competitive facility location problem which deals with the favorable placement of facilities by competing market players. Facilities and users are generally modeled as points in the plane. The set of users (demands) is either discrete, consisting of finitely many points, or continuous, i.e., a region where every point is considered to be a user. We assume that the facilities are equally equipped in all respects, and a user always avails the service from its nearest facility. Given a set of facilities F, service zone of a facility $f \in F$, $U(f, F)$ is the set of points in the user space that are closer to f than any other facility in $f' \in F$. The term Voronoi Game was first introduced by Ahn et al.[1]. They have considered the game where the users are distributed in a unit length line segment ℓ such that each point on the line segment ℓ is a user and two players, Player 1(P1) and Player 2(P2) will sequentially place a set of k facilities F and S respectively in the line segment. Payoff of P1 is defined as $\cup_{f \in F} U(f, F \cup S)$. They showed that the second player always has a winning strategy that guarantees a payoff of $1/2 + \epsilon$, with $\epsilon \geq 0$. However, the first player can force ϵ to be arbitrarily small. When the user space is a subset of $\mathbb{R}^2$, the problem reduces to maximizing the area of the Voronoi regions of

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 210–220, 2013.

the set of facilities. Dehne et al. [10] addressed the problem of finding a new point q amidst a set of n existing points F such that the Voronoi region of q is maximized, and studied the case when the points in F, the set of facilities are in convex position. For the same problem, Cheong et al. [8] gave a near-linear time algorithm that locates the new optimal point approximately, when the points in F are in general position. Cheong et al. [9], considered the case when the user space is a square. They proved that for any placement W of the first player, with $|W| = m$, there is a placement B of the second player $|B| = m$ such that the payoff of the second player is at least $1/2 + \alpha$, where $\alpha > 0$ is an absolute constant and m large enough. Fekete and Meijer [11] studied the two-dimensional one-round game played on a rectangular demand region with aspect ratio ρ. A variation of this problem, involving maximization of the area of Voronoi regions of a set of points placed inside a circle, has been considered by Bhattacharya et al. [5]. The Voronoi game when the demand region is a graph is considered in the paper by Bandyapadhyay et al.[2].

In the discrete user case, the analogous problem is to place a set of new facilities amidst a set of existing ones such that the number of users served by the new facilities is maximized. The problem of placing only one new facility has been addressed by Cabello et al. [7] and is referred to as the $MaxCov$ problem. The $2-MaxCov$ problem, which considers the problem of placing two new facilities, has been studied by Bhattacharya and Nandy [6]. Teramoto et al. [12] studied the same problem and considered following very restricted case: the game arena is an arbitrary graph, the first player occupies just one vertex which is predetermined, and the second player occupies m vertices in any way. They proved that in this strongly restricted discrete Voronoi game it is NP-hard to decide whether the second player has a winning strategy. They also proved that for a given graph G and the number r of rounds determining whether the first player has a winning strategy on G is PSPACE-complete.

Banik et al. [3] studied the one-round discrete Voronoi game on $\mathbb{R}$, and gave algorithms for obtaining optimal strategies of the two players. The problem consists of a finite user set $U \subset \mathbb{R}$, with $|U| = n$, and two players Player 1 (P1) and Player 2 (P2) each having $m = O(1)$ facilities. At first, P1 chooses a set $\mathcal{F}_1 \subset \mathbb{R}$ of m facilities following which P2 chooses another set $\mathcal{F}_2 \subset \mathbb{R}$ of m facilities, disjoint from $\mathcal{F}_1$. The payoff of P2 is defined as the cardinality of the set of points in U which are closer to a facility owned by P2 than to every facility owned by P1. The payoff of P1 is the number of users in U minus the payoff of P2. The objective of both the players is to maximize their respective payoffs. The authors showed that if the sorted order of the points in U along the line is known, then the optimal strategy of P2, given any placement of facilities by P1, can be computed in $O(n)$ time. Also, for $m \geq 2$ the optimal strategy of P1 can be computed in $O(n^{m-\lambda_m})$ time, where $0 < \lambda_m < 1$, is a constant depending only on m. Note that unlike the Voronoi game on a graph, the players can choose their facilities anywhere on the real line. Moreover, as the distances between the facilities and users are measured in the Euclidean metric, the geometric locations

of the users in U on the real line is crucial in the discrete Voronoi game on $\mathbb{R}$. Discrete Voronoi game in polygonal domain has been studied in [4].

In this paper, we shall study the k-round discrete Voronoi game on a line. The game consists of a set U of n users or demands on a line ℓ and two players $P1$ and $P2$. Both the players will place a set of k facilities on the line ℓ sequentially, i.e. initially $P1$ will place one facility followed by which $P2$ will place another facility and this will continue for k rounds. At the end of the game, given the placement of k facilities F and S by P1 and P2 respectively we define the payoff of P1, $Q_1(F, S)$ to be equal to $|\cup_{f\in F} U(f, F \cup S)|$. Similarly we define payoff of P2, $Q_2(F, S)$ to be equal to $|U| - Q_1(F, S)$. Any k round game will contain $2k$ many moves.

For example, When $k = 2$, the optimal placement at fourth move, given f_1, s_1, f_2 is the point which minimizes $Q_1(\{f_1, f_2\}, \{s_1, x\})$ over all $x \in \ell$, and optimal placement at third move given f_1, s_1 is a point which maximizes $\min_{x\in\ell} Q_1(\{f_1, y\}, \{s_1, x\})$ over all $y \in \ell$ and so on. Observe now when $k = 1$, optimal placement of $P1$ is at the median of the set of users U. In this paper, we have considered the first non-trivial case when $k = 2$.

2 Lower Bounds

Theorem 1. *In the two-round discrete Voronoi game, $P2$ always gets at least $n/2$ users, i.e, $P2$ always wins and this bound is tight. Furthermore, there exists a strategy such that $P1$ always gets at least $n/3$ users irrespective of the placement of $P2$.*

Fig. 1. Placement by P2 in different situations to ensure payoff equals to $n/2$

Proof. We first show that there exists a placement strategy for $P2$ such that $Q_2(F, S) \geq \lfloor \frac{n}{2} \rfloor$ users irrespective of the placement of $P1$'s facilities. Observe that for any placement f_1 by $P1$ either $|(-\infty, f_1] \cap U| \leq n/2$ or $|[f_1, \infty) \cap U| \leq n/2$. Without loss of generality, we assume that, $|(-\infty, f_1] \cap U| = k \leq n/2$. Observe that, there exists a placement s_1 by $P2$ such that $|(f_1, s_1) \cap U| = n/2 - 1$ and s_1 coincides with an user (see Figure 1(a)). Now for any placement f_2 by $P1$ either $f_2 \in (-\infty, f_1)$ or $f_2 \in (f_1, s_1)$ or $f_2 \in (s_1, \infty)$.

Case1: If $f_2 \in (-\infty, f_1)$, then for optimal placement of s_2, $|(f_1, s_2) \cap U| = 0$. Now, since $k \leq n/2$, for this case $Q_2(F, S) \geq n/2$ (see Figure 1(b)).

Case 2: If $f_2 \in (f_1, s_1)$, then the optimal placement of s_2 by $P2$, will be such that $|(s_2, f_1) \cap U| = 0$ resulting in the minimum payoff of $(n - |[f_1, f_2] \cap U|) \geq n/2$ (see Figure 7(a)).

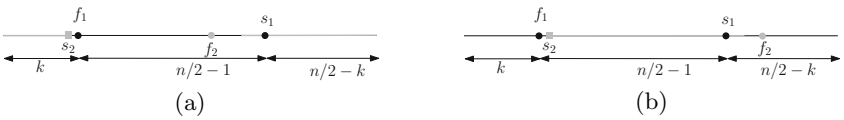


Fig. 2. Placement by P2 in different situations to ensure payoff equals to $n/2$

Case 3: If $f_2 \in (s_1, \infty)$, then the optimal placement of s_2 by $P2$ will be such that $|(f_1, s_2) \cap U| = 0$. So, here $Q_2(F, S) \geq n/2$ (see Figure 7(b)).

So, $P2$ always get $n/2$ number of users. Now, our claim is, if the users in the set U are uniformly distributed along line ℓ, there exists a strategy by $P1$ for which $P2$ cannot get more than $n/2$ users. $P1$ places f_1 such that $|(-\infty, f_1] \cap U| = n/4$. Now if s_1 places such that $|[f_1, s_1] \cap U| < n/2$, f_2 can place in (s_1, ∞) such that $|[f_2, \infty) \cap U| = n/4$. Then it can be checked that even for the optimal placement of s_2, $P2$ gets less than $n/2$ users. If $|[f_1, s_1] \cap U| > n/2$, f_2 places in (f_1, s_1) such that $|[f_2, \infty) \cap U| = n/4$. It can be verified that, in that case $P2$ gets less than $n/2$ users. And for $|[f_1, s_1] \cap U| = n/2$, for any placement of f_2, if s_2 is placed optimally, $P2$ gets $n/2$ users. So, for all the cases, $P2$ cannot get more than $n/2$ users.

Now we will prove the second part of the theorem. Note that if $|(-\infty, f_1] \cap U| = n/3$ and $|(-\infty, s_1] \cap U| \neq 2n/3$ then f_2 must place such that $|(-\infty, f_2] \cap U| = 2n/3$. On the other hand if $|(-\infty, s_1] \cap U| = 2n/3$, then f_2 places such that $|(s_1, f_2) \cap U| = 0$, i.e., on the $(2n/3 + 1)^{th}$ user. Now, it can be easily observed that f_1 and f_2 divides l in three intervals each of which has length of either $n/3$ or $(n/3-1)$. $P2$ can take the users of at most two intervals by placing 2 facilities. All the users of one of these 3 intervals will always belong to $P1$. So, following this strategy $P1$ will always get at least $n/3$ users. □

3 Optimal Strategies of $P1$ and $P2$ While Placing Second Facilities

In this section, given two placement of facilities f_1 and s_1 placed by P1 and P2 respectively, we will provide an strategy to find the optimal placements by P1 and P2 while placing their second facilities. Initially we will find an optimal placement for P2 given the placement of facilities $\{f_1, f_2\}$ by P1 and $\{s_1\}$ by P2. That is we want to find an placement which maximizes $Q_2(\{f_1, f_2\}, \{s_1, x\})$ over all placement of facility x by P2. Next we will move dipper into the game and given two placement of facilities f_1 and s_1 by P1 and P2 we will find a point which minimizes $\max_{x \in \ell} Q_2(\{f_1, y\}, \{s_1, x\})$ over all placement of facility y by P1.

Let us begin our discussion with some general results about discrete Voronoi game in a line segment most of which we have borrowed from [3]. Any set of facilities placed by two players divides ℓ into a set of line segments. Placement of a new facility in one of such intervals does not affect the users in the other intervals. Now end point of each such intervals may be of three types, bounded

by two facilities placed by the same player, bounded by two facilities placed by different players or bounded by one facility. Observe that in later two types of intervals by placing one facility any player can serve all the users in that interval. For the other type of interval let us define a notation $cov(x, y)$(cover) which will be useful throughout the paper.

Definition 1. *For two facilities f_a and f_b, by P1 (resp. P2) such that there is no facility in the interval $[f_a, f_b]$, $cov([f_a, f_b])$ is the maximum number of users that can be served by a single facility placed by P2 (resp. P1).*

Next observation is a direct consequence of the paper [3].

Observation 1. *Given the placement of facilities f_1 and f_2 such that $s_1 \notin [f_1, f_2]$, maximum number of facilities that P2 can serve by placing one facility in the interval $[f_1, f_2]$ can be obtained in $O(n)$ time.*

Now we have the following lemma.

Lemma 1. *The optimal strategy for placing s_2 by P2 can be obtained in $O(n)$ time.*

Proof. The optimal placement of s_2 in the intervals $(-\infty, f_1)$ and (f_1, s_1) or (f_2, s_1) or (s_1, f_2) can be found in $O(1)$ time. From Observation 1, the optimal placement of s_2 in the interval (f_1, f_2) or (f_2, f_1) can be obtained in $O(n)$ time. Hence the result follows. □

Next we will provide an algorithm to find the optimal placement by P1, while placing the second facility given the placement of facilities f_1 and s_1 by P1 and P2 respectively. But before that that consider the following definitions.

Definition 2. *For some positive integer k and $x \in \ell$, $g_x(k) \in \ell$ is the maximum point such that $cov([x, g_x(k)]) = k$ if such a point exists. We also define, $h_x(k)$ to be minimum point such that $cov([h_x(k), x]) = k$.*

For each user $u_i \in U$, consider the set of points

$$D_i = \{x : x = s_1 \pm 2|s_1 - u_i|\}$$

Let $D = \cup_{i \le i \le n} D_i$. Now consider the following lemma.

Lemma 2. *One of the optimal placements by P1 belongs to $U \cup D$ while placing the second facility.*

Proof. For any interval $[a, b]$ on ℓ we will denote the users in the interval $[a, b]$ by $U[a, b]$. Observe that there may be two cases, either $f_1 < s_1$ or vice verse. We will assume that $f_1 < s_1$ the other case can be dealt similarly. Let f_2 be any optimal placement by P1 which does not coincides with any point on $U \cup D$. Now f_2 can belong to one of the three intervals, $[-\infty, f_1]$, $[f_1, s_1]$ or $[s_1, \infty]$. In this paper we will only prove the case when $f_2 \in [f_1, s_1]$. The other two cases can be proved similarly. Let $p < f_2$ be the point closest to f_2 in the set of points $U \cup D$. Now observe for the placement of facility f_2 payoff of P1 is equals to

$$\{|U[-\infty, f_2] + U[f_2, (f_2 + s_1)/2|\} - \{\max(|U[-\infty, f_1]|, cov(f_1, f_2), |U[f_2, (f_2 + s_1)/2]|)\}$$

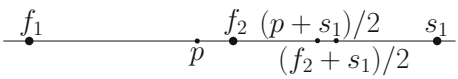


Fig. 3. Illustration of Lemma 2

Observe now $cov(f_1, f_2) < cov(f_1, p)$ and $U[f_2, (f_2 + s_1)/2 = U[p, (p + s_1)/2$ because $[(f_2 + s_1)/2, (p + s_1)/2]$ does not contain any user as $p < f_2$ be the point closest to f_2 in the set of points $U \cup D$. therefore payoff of P1 when places second facility at f_2 is less than equals to the payoff of P1 when places second facility at p. Hence the result holds.

Observe that cardinality of $U \cup D$ is $O(n)$. Hence by checking each point we can find out the optimal placement by P1 while placing the first facility. Hence we have the following theorem.

Theorem 2. *The optimal strategy of $P1$ while placing the first facility can be found in $O(n^2)$ time.*

4 Optimal Strategy of P2 While Placing the First Facility

In this section we will provide a strategy to find an optimal placement by P2 while placing the first facility, that is, given a placement of facility f_1 by P1, we want to find a point which maximizes $\min_{y \in \ell} \max_{x \in \ell} Q_2(\{f_1, y\}, \{z, x\})$ over all point $z \in \ell$.

Now $P2$ can place its first facility either in $(-\infty, f_1)$ or (f_1, ∞). Here, we provide an algorithm where $P2$ is restricted to place its first facility, s_1 in the interval (f_1, ∞). Other case can be dealt similarly. Given f_1, for any point $s_1 \in \ell$ let us define

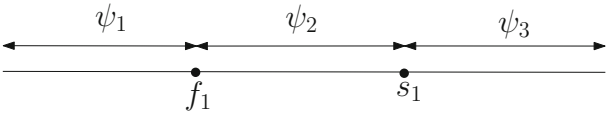


Fig. 4. Domain of three functions ψ_1, ψ_2 and ψ_3

$$\psi(s_1) = \min_{f_2 \in \ell} \max_{s_2 \in \ell} Q_2(\{f_1, f_2\}, \{s_1, s_2\})$$

Given f_1 and s_1 P1 can place it's second facility in one of the three intervals $[-\infty, f_1]$, $[f_1, s_1]$ or $[s_1, \infty]$. Given two placement of facilities $\{x, z\}$ by P1 and $\{y\}$ by P2 denote $\max_{p \in \ell} Q_2(\{x, y\}, \{z, p\}$ by $Q(x, y, z)$. Hence we have,

$$\psi(s_1) = \min \Big[\big(\min_{f_2 \in [-\infty, f_1]} Q(f_1, s_1, f_2) \big), \big(\min_{f_2 \in [f_1, s_1]} Q(f_1, s_1, f_2) \big), \big(\min_{f_2 \in [s_1, \infty]} Q(f_1, s_1, f_2) \big) \Big]$$

Now we can write,

$$\psi(s_1) = \min(\psi_1(s_1), \psi_2(s_1), \psi_3(s_1))$$

where $\psi_1(s_1) = \min_{f_2 \in [-\infty, f_1]} Q(f_1, s_1, f_2))$, $\psi_2(z) = \min_{f_2 \in [f_1, s_1]} Q(f_1, s_1, f_2)$ and $\psi_3(z) = \min_{f_2 \in [s_1, \infty]} Q(f_1, s_1, f_2))$.

In other words $\psi_1(s_1), \psi_2(s_1)$ and $\psi_3(s_1)$ denotes, when first facility of P2, s_1 moves along the line ℓ how the payoff of P2 changes when P1 is restricted to place it's second facility at intervals $[-\infty, f_1], [f_1, s_1]$ and $[s_1, \infty]$ respectively. Observe that ψ, ψ_1, ψ_2 and ψ_3 can take $n+1$ many discrete values, $\{0, 1 \ldots n\}$. Hence there are a certain set of points in ℓ in which each of ψ_1, ψ_2, ψ_3 jumps from one value to another. By calculating the payoff at each such point we can find out the optimal placement of first facility by $P2$. We call such points as event points.

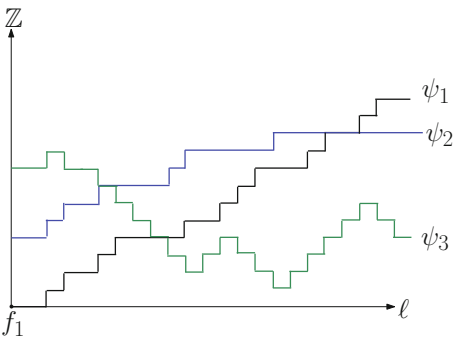


Fig. 5. ψ_1, ψ_2, ψ_3

Lemma 3. *There are $O(n)$ number of event points in ψ_1 which can be computed in $O(n)$ time.*

Proof. The set of points where $\psi_1(s_1)$ changes can be defined by the set $A_2 \cup B_2$, where

$$A_2 = \{a | a = f_1 + 2|u - f_1|, u \in [f_1, \infty) \cap U\}$$
$$B_2 = \{u | u \in (f_1, \infty) \cap U\}$$

The points of set A_2 are the points where if s_1 is placed, $(f_1 + s_1)/2$ is on a user. So, at this position of s_1, $|[f_1, (f_1 + s_1)/2] \cap U|$ changes, i.e., $\psi_1(s_1) = n - (|[s_1, \infty] \cap U| + max(\delta, |[f_1, (f_1 + s_1)/2) \cap U|])$ changes, where $\delta = max(|[-\infty, f_2] \cap U|, cov([f_2, f_1]))$. The points of B_2 are the points where s_1 is on a user. So, $|[s_1, \infty) \cap U|$ changes and thus $\psi_1(s_1)$ also changes. It is easy to observe that the points of set A_2 and B_2 can be found in $O(n)$ time. □

Lemma 4. *There are $O(n^2)$ number of event points in ψ_2 which can be computed in $O(n^2)$ time.*

Proof. We know that, if $g_{f_1}(\alpha) < s_1$ the set of points which can be considered as candidates for optimal placement of f_2 can be expressed as [From Lemma 3.4.]:

$$C = \{x | x = g_{f_1}(i) \wedge x \leq s_1, i \geq \alpha\}$$

Now for $f_2 \in C$, when $|[(f_2 + s_1)/2, s_1] \cap U|$ changes, i.e. when $(f_2 + s_1)/2 \in U$, ψ_2 changes. So, the set of points at which ψ_2 changes, i.e. the set of event points, can be expressed as:

$$A_1 = \{a | a = u + |u - x|, x \in C, u \in [x, s_1] \cap U\}$$

Now, the points in set C can be found in $O(n^2)$ time. There are $O(n)$ points in C. It is easy to observe that, for each point in C, there are $O(n)$ points in set A_1, which can be computed in $O(n)$ time. So, all the points in set A_1 can be found in $O(n^2)$ time. If $g_{f_1}(\alpha) > s_1$, it can be easily observed the set of users U is the set of event points. So, all the event points can be computed in $O(n^2)$ time. □

Lemma 5. *There are $O(n^2)$ number of event points in ψ_3 which can be computed in $O(n^2)$ time.*

Proof. Say, for some placement of s_1 at a point $x \in (f_1, \infty)$ when $f_2 \in (s_1, \infty)$, $y > x$ is the minimum point such that $\psi_3(y) \neq \psi_3(x)$. Then, either $(f_1+y)/2 \in U$ or $(y + f_2)/2 \in U$. We also know that for the optimal placement of f_2, $f_2 \in U$. So, the set of positions of s_1, where ψ_3 changes can be defined by $A_3 \cup B_3$ where, $A_3 = \{a | a = f_1+2|u-f_1|, u \in [f_1, \infty) \cap U\}$ and $B_3 = \{b | b = u_i - |u_j - u_i|, u_i, u_j \in [f_1, \infty) \cap U, u_i < u_j \forall i, j < n\}$ The points in set A_3 are the points where if s_1 is placed, $(f_1 + s_1)/2$ is on a user. We know from Lemma 3.5., $f_2 \in U$ when $f_2 \in (s_1, \infty)$. The set B_3 represents the set of points where if s_1 is placed, for a placement of f_2 on user, $(s_1 + f_2)/2$ is on user. The points of set A_3 can be found in $O(n)$ time and the points of set B_3 can be found in $O(n^2)$ time. □

Theorem 3. *The optimal placement of the first facility of $P2$, s_1 can be found in $O(n^2)$ time.*

Proof. We know from Lemma 3, Lemma 4 and Lemma 5 we know that the event points in ψ_1, ψ_2, ψ_3 can be found in $O(n)$, $O(n^2)$ and $O(n^2)$ time respectively. Each of ψ_1, ψ_2, ψ_3 either increases or decreases by unit amount at their event points. So, the values of ψ_1, ψ_2 and ψ_3 at the event points can also be found in $O(n)$, $O(n^2)$ and $O(n^2)$ time respectively by a linear scan of the event points. Having the sorted order of the event points of ψ_1, ψ_2, ψ_3 and knowing the values of ψ_1, ψ_2, ψ_3 at those points, the upper envelope, the minimum of the upper envelope of ψ_1, ψ_2, ψ_3 and the payoff of $P1$ at that point can be found in $O(n^2)$ time. □

5 Optimal Strategy of P1 While Placing the First Facility

The optimal placement of the first facility by $P1$ should minimize the maximum payoff of $P2$ considering s_1, s_2 and f_2 will be placed at optimal position in the

next rounds. f_1, s_1, f_2, s_2 can permute on real line ℓ in 24 ways. For each of these cases, for any placement of first facility of $P1$, $x \in \ell$ we define a function $\eta_i(x)$ $(\forall i,\ 1 \leq i \leq 24)$ to be the minimum number of users that $P1$ can occupy for permutation i. Now observe that the maxima of the lower envelope of the functions η_i, $\forall i,\ 1 \leq i \leq 24$, corresponds to an optimal placement of f_1 .

We observe that $\eta_i (1 \leq i \leq 24)$ can take at most $n+1$ many discrete values(0 to n). Hence there are a certain set of points in ℓ in which each of η_i jumps from one value to another. We call these points event points. Observe that, there exists an optimal placement of f_1 by $P1$ coincides with one of the event points. In this section we will prove that the number of event points are polynomial in the input size and we can find out the set of all event points in polynomial time. Finally we will show that by computing the payoff of $P1$ at each event point, we can find out an optimal placement of f_1 by $P1$ in polynomial time. The set of arrangements for which $s_1 \in (f_1, \infty)$ is symmetrically opposite to the set of permutations for which $s_1 \in (-\infty, f_1)$. Hence, it is enough to show that the functions corresponding to the permutations for which $s_1 \in (f_1, \infty)$ contain polynomial many event points. The other case follows from symmetry.

In this paper, we will find out the set of event points, assuming that $P2$ will place its first facility in the interval (f_1, ∞). The set of event points when $P2$ is restricted to place its first facility in the interval $(-\infty, f_1)$ can be computed similarly.

Lemma 6. *If f_2 is restricted to place in $(-\infty, f_1)$ and $s_1 \in (f_1, \infty)$, there exists an optimal placement of s_1 which coincides with a user.*

Proof. We assume, there exists an optimal placement x of s_1 in (f_1, ∞) such that $x \notin U$ and $x \in (u_i, u_j)$ where $u_i, u_j \in U$ and there is no user in (u_i, u_j). Now if s_1 places at u_i instead of x, number of users in (s_1, ∞) reamins same, $(f_1 + s_1)/2$ moves left, number of users in $[(f_1 + s_1)/2, s_1]$ may increase, which may lead to f_2 placing in (f_1, s_1). In that case, $|[(f_1 + x)/2, x] \cap U| < \max(|[f_2, f_1] \cap U|, |(-\infty, f_2] \cap U|) < |[(f_1 + u_i)/2, u_i] \cap U|$. But, it is to observe that even when s_1 is placed at x the users in $[(f_1 + u_i)/2, (f_1 + x)/2]$ were acquired by $P1$, so number of user $P2$ occupies when placed at u_i is not less than the number of userss it occupies when placed at x. So, s_1 is placed on one of the users. □

Theorem 4. *There exists an optimal placement f_1' by $P1$ such that $f_1' \in \Gamma$ where $\Gamma = \{a | a = u_i \pm 2|u_j - u_k| \forall u_i, u_j, u_k \in U, j \neq k \vee a \in U\}$.*

Proof. Considering $s_1 \in (f_1, \infty)$, there are 12 ways to arrange f_1, s_1, f_2, s_2 on ℓ. We will find out the set of event points for each of these 12 cases separately. Among these 12 cases, there are 2 cases which never occur. It can be easily observed that s_2 never places in (s_1, ∞). So there is no need to consider the cases $f_1 < f_2 < s_1 < s_2$ and $f_2 < f_1 < s_1 < s_2$. Among the other cases we will present 3 cases, and omit the details of other cases for space restriction.

Case 1: $f_2 < s_2 < f_1 < s_1$

It is to observe that in this case payoff of $P1$ changes when either of $cov([f_2, f_1])$ and $|[f_1, (f_1 + s_1)/2] \cap U|$ changes. We can observe that when $(f_1 + s_1)/2$ is

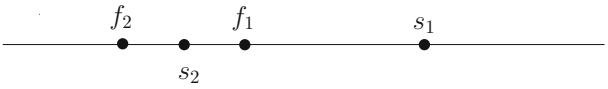


Fig. 6. $f_2 < s_2 < f_1 < s_1$ (Case 1)

on a user, $|[f_1, (f_1 + s_1)/2] \cap U|$ changes. Also s_1 is on a user from Lemma 5.1. So, the positions of f_1 for which $|[f_1, (f_1 + s_1)/2] \cap U|$ changes can be expressed by the following set:

$$A_1 = \{a | a = u_k - 2|u_j - u_k|, u_j, u_k \in U, u_j < u_k\}$$

Also we know that for each possible value of $cov([f_2, f_1])$, there is an interval $[u_k, u_l]$, $u_k, u_l \in U$ such that there are $cov([f_2, f_1])$ users in the interval $[u_k, u_l]$. Also, f_2 is on a user from Lemma 3.2.. So, considering each possible of placement of f_2 on user and all pair of users u_k, u_l, we get the set of event points that can be expressed as:

$$A_2 = \{b | b = u_k + 2|u_i - u_j|, u_k, u_i, u_j \in U, i \neq j \neq k, 1 \leq i, j, k \leq n\}$$

We can observe that $A_1 \cup A_2 \subset \Gamma$.

Case 2: $f_1 < s_1 < f_2 < s_2$

From Lemma 3.5., f_2 is placed on a user. Now, it is to observe that irrespective of the interval where s_2 places in, there is an interval $[u_i, u_j]$ (where $u_i, u_j \in U \cap [f_1, f_2]$) such that all the users in the interval $[u_i, u_j]$ are captured by s_1. So, if we consider all possible u_i, u_j pairs and all possible placements of f_2, we have a a set of breakpoints that can be expressed as

$$A_3 = \{b | b = u_k - 2|u_i - u_j|, u_k, u_i, u_j \in U, i \neq j \neq k, 1 \leq i, j, k \leq n\}$$

It is evident that $A_3 \subset \Gamma$.

Fig. 7. (a)Case 2 (b)Case 3

Case 3: $f_1 < s_2 < f_2 < s_1$

Now for some placement of $s_2 \in [f_1, f_2]$, s_2 occupies the users of an interval $[u_i, u_j]$ for some $u_i, u_j \in U$ where $|f_1 - f_2| = 2|u_i - u_j|$. We imagine that an interval $[f_1, f_2]$ of length $2|u_i - u_j|$ is moving left to right along ℓ. When f_2 is on u_j, $[u_i, u_j]$ becomes a subinterval of $[f_1, f_2]$ for the first time and thus the payoff of $P1$ changes. Again as the interval $[f_1, f_2]$ moves, when f_2 crosses any user, payoff of $P1$ changes again. Now, when f_1 is on u_j the interval $[u_i, u_j]$ is no longer a subinterval of $[f_1, f_2]$ and the number of users s_2 can occupy from the interval $[f_1, f_2]$ changes. Now, if we consider all such user pairs u_i, u_j, we get the set of breakpoints which can be expressed as:

$$A_4 = U \cup \{b | b = u_k - 2|u_i - u_j|, u_k, u_i, u_j \in U, u_i < u_j, i \neq j \neq k, 1 \leq i, j, k \leq n\}$$

Clearly, $A_4 \subset \Gamma$.

Clearly, for the above cases, one of the event points in Γ is an optimal placement of f_1. For the remaining other cases also, it can be shown that one of the event points in Γ is an optimal placement of f_1. □

The points of the set Γ can be found in $O(n^3)$ time. For each event point in Γ, optimal strategy of s_1 and the corresponding payoff of $P1$, considering f_2 and s_2 will place optimally in the next round can be computed in $O(n^2)$ time by Theorem 5.1. So, maxima of lower envelope of η_i $\forall i$ such that $1 \leq i \leq 24$ can be found in $O(n^5)$ time, which corresponds to an optimal placement of f_1. So we can conclude the following theorem:

Theorem 5. *The optimal placement of the first facility of $P1$, f_1 can be found in $O(n^5)$ time.*

References

1. Ahn, H.-K., Cheng, S.-W., Cheong, O., Golin, M.J., van Oostrum, R.: Competitive facility location: the Voronoi game. Theor. Comput. Sci. 310(1-3), 457–467 (2004)
2. Bandyapadhyay, S., Banik, A., Das, S., Sarkar, H.: Voronoi Game on Graphs. In: Ghosh, S.K., Tokuyama, T. (eds.) WALCOM 2013. LNCS, vol. 7748, pp. 77–88. Springer, Heidelberg (2013)
3. Banik, A., Bhattacharya, B.B., Das, S.: Optimal strategies for the one-round discrete Voronoi game on a line. Journal of Combinatorial Optimization, 1–15 (2012)
4. Banik, A., Das, S., Maheshwari, A., Smid, M.: The discrete voronoi game in a simple polygon. In: Du, D.-Z., Zhang, G. (eds.) COCOON 2013. LNCS, vol. 7936, pp. 197–207. Springer, Heidelberg (2013)
5. Bhattacharya, B.B.: Maximizing Voronoi regions of a set of points enclosed in a circle with applications to facility location. J. Math. Model. Algorithms 9(4), 375–392 (2010)
6. Bhattacharya, B.B., Nandy, S.C.: New variations of the maximum coverage facility location problem. European Journal of Operational Research 224(3), 477–485 (2013)
7. Cabello, S., Díaz-Báñez, J.M., Langerman, S., Seara, C., Ventura, I.: Facility location problems in the plane based on reverse nearest neighbor queries. European Journal of Operational Research 202(1), 99–106 (2010)
8. Cheong, O., Efrat, A., Har-Peled, S.: On finding a guard that sees most and a shop that sells most. In: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, pp. 1098–1107. Society for Industrial and Applied Mathematics, Philadelphia (2004)
9. Cheong, O., Linial, N., Har-peled, S.: The one-round Voronoi game. Discrete Comput. Geom., 97–101 (2002)
10. Dehne, F., Klein, R., Seidel, R.: Maximizing a Voronoi region: The convex case. In: Proc. 13th Annu. Internat. Sympos. Algorithms Comput., pp. 624–634 (2005)
11. Fekete, S.P., Meijer, H.: The one-round Voronoi game replayed. Comput. Geom. 30(2), 81–94 (2005)
12. Teramoto, S., Demaine, E.D., Uehara, R.: Voronoi game on graphs and its complexity. In: Louis, S.J., Kendall, G. (eds.) CIG, pp. 265–271. IEEE (2006)

Inverse Maximum Flow Problems under the Combining Norms

Longcheng Liu*

School of Mathematical Sciences, Xiamen University, Xiamen 361005, P.R. China
longchengliu@xmu.edu.cn

Abstract. Given a network $N(V, A, c)$ and a feasible flow x^0, an inverse maximum flow problem is to modify the capacity vector as little as possible to make x^0 form a maximum flow of the network. The modification can be measured by different norms. In this paper, we consider the inverse maximum flow problems under the combining norms, i.e., the modification cost is fixed in a given interval, and is depended on the modification out of the given interval. For both sum-type and bottleneck-type cases, we present their respective combinatorial algorithms that all run in strongly polynomial times.

Keywords: Maximum flow, Inverse problem, Combining norms, Strongly polynomial algorithm.

1 Introduction

Let $N(V, A, c)$ be a connected and directed network, where $V = \{1, 2, \ldots, n\}$ is the node set, A is the arc set ($|A| = m$) and c is the capacity vector for arcs. Each component c_{ij} of c is called the *capacity* of arc (i, j). There are two special nodes in V: the source node s and the sink node t. An (s, t)-*flow* or simply *flow* is a function $f : A \to R^{|A|}$ satisfying the capacity constraints

$$0 \le f_{ij} \le c_{ij} \quad \text{for each } (i, j) \in A,$$

and the flow conservation constraints

$$\sum_j f_{ij} - \sum_j f_{ji} = \begin{cases} v(f), & i = s, \\ -v(f), & i = t, \\ 0, & \text{other } i, \end{cases}$$

where $v(f)$ is the value of flow f from s to t. The maximum flow problem is to find a flow with the maximum flow value. It is a classical network optimization problem that has many applications. It is well known that maximum flow problem can be solved in strongly polynomial time.

* This research is supported by the National Natural Science Foundation of China (Grant No. 11001232), Fujian Provincial Natural Science Foundation of China (Grant No. 2012J01021) and Fundamental Research Funds for the Central Universities (Grant No. 2010121004).

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 221–230, 2013.

Conversely, an inverse maximum flow problem is to modify the arc capacity vector as little as possible such that a given flow can form a maximum flow. Without loss of generality, an inverse maximum flow problem can be stated as follows:

$$\begin{aligned} &\min C(d-c) \\ &\text{s.t. Flow } f^0 \text{ is a maximum flow of } N(V,A,d). \end{aligned} \tag{1}$$

Where f^0 is a given feasible flow in the network $N(V,A,c)$, and $C(d-c)$ is a (weighted) deviation of vector d from vector c, or the cost of changing c into d. [9] showed that the inverse maximum flow problem under l_1 norm is strongly polynomial time solvable, where the modification cost is measured by l_1 norm. For the inverse maximum flow problems under the weighted Hamming distance, [7] presented strongly polynomial algorithms for both sum-type and bottleneck-type cases. In this paper, we consider the inverse maximum flow problem under the combining norms, in which we measure the modification cost by the combining norms, i.e., the modification cost is fixed in a given interval, and is depended on the modification out of the given interval. We focus on two models.

Let each arc (i,j) have an associated capacity modification cost such that: if the new capacity d_{ij} is in a given interval $[p_{ij},q_{ij}]$, i.e., $p_{ij} \le d_{ij} \ne c_{ij} \le q_{ij}$, then the modification cost is fixed, say as w_{ij}. Otherwise, the modification cost is linearly depended on the modification, i.e., if $d_{ij} < p_{ij}$, then the modification cost is $w_{ij}+k_{ij}\cdot(p_{ij}-d_{ij})$, if $d_{ij} > q_{ij}$, then the modification cost is $w_{ij}+k_{ij}\cdot(d_{ij}-q_{ij})$. Let f^0 be a given feasible flow in the network $N(V,A,c)$. Then for the inverse maximum flow problem under the sum-type combining norms, we look for an arc capacity vector d such that

(a) f^0 is a maximum flow of network $N(V,A,d)$;

(b) for each $(i,j) \in A$, $-l_{ij} \le d_{ij}-c_{ij} \le u_{ij}$, where $l_{ij},u_{ij} \ge 0$ are respectively given bounds for decreasing and increasing capacity c_{ij};

(c) the total modification cost for changing capacities of all arcs, i.e., $\sum_{(i,j)\in A}\{w_{ij}H(c_{ij},d_{ij}) + k_{ij}\cdot\max\{p_{ij}-d_{ij},0,d_{ij}-q_{ij}\}\}$, is minimized, where $H(c_{ij},d_{ij})$ is the Hamming distance between c_{ij} and d_{ij}, i.e., $H(c_{ij},d_{ij})=0$ if $c_{ij}=d_{ij}$ and 1 otherwise.

For the inverse maximum flow problem under the bottleneck-type combining norms, we look for an arc capacity vector d such that, under the constraints (a) and (b),

(c′) the maximum modification cost among all arcs, i.e., $\max_{(i,j)\in A}\{w_{ij}H(c_{ij},d_{ij})+ k_{ij}\cdot\max\{p_{ij}-d_{ij},0,d_{ij}-q_{ij}\}\}$, is minimized.

In general, in an inverse combinatorial optimization problem, a feasible solution is given which is not optimal under the current parameter values, and it is required to modify some parameters with minimum modification cost such that the given feasible solution becomes an optimal solution. A lot of such problems have been well studied when the modification cost is measured by (weighted) l_1, l_2, l_∞ norms and Hamming distance. Some examples are the inverse shortest path problem [2] [11], the inverse minimum spanning tree problem [10] [3] [5],

the inverse minimum cut problem [9] [6] and so on. For more detail, readers may refer to the survey paper [4] and papers cited therein.

The problems considered in our paper has so far not been treated in literature, but seems to have some potential applications in real world. For example, in practice, we often wish to reduce the traveling time (or increase the runoff) through a road by widening the road. At the beginning of the project, we should make some design for the project and buy some mechanical equipment. Those costs are fixed for the project. As the project progresses, the new costs are depend on the modification. Another example, with the development of the computer network, users' demand on the network changes persistently. More kinds of information transmitted and higher quality of services (Qos) provided are greatly required. To meet those requirements, we need to modify the exist network. At the beginning, we should buy some high-performance server, which can be seen as fixed cost. And during modify the network, the new costs may depend on the modification. So it is meaningful to consider the inverse optimization problems under the combining norms.

The remainder of the paper is organized as follows. Section 2 considers the inverse maximum flow problem under the sum-type combining norms. Sections 3 considers the inverse maximum flow problem under the bottleneck-type combining norms. We show that all these problems can be solved by strongly polynomial algorithms. Some final remarks are made in Section 4.

2 Inverse Maximum Flow Problem under the Sum-Type Combining Norms

Let X and $\overline{X} = V \setminus X$ be a partition of all vertices such that $s \in X$ and $t \in \overline{X}$. An $s-t$ cut, denoted by $\{X, \overline{X}\}$, is the set of arcs with one endpoint in X and another endpoint in $\overline{X}$. We further use $(X, \overline{X})$ to express the set of forward arcs from a vertex in X to a vertex in $\overline{X}$ and use $(\overline{X}, X)$ to express the set of all backward arcs in the $s-t$ cut. As we know the capacity of the $s-t$ cut $\{X, \overline{X}\}$, denoted by $C(\{X, \overline{X}\})$, is the sum of the capacities of all forward arcs:

$$C(\{X, \overline{X}\}) = \sum_{(i,j)\in(X,\overline{X})} c_{ij}.$$

The following result is well known.

Lemma 1. *[1] A feasible flow f of the network $N(V, A, c)$ is the maximum flow if and only if the value of flow f equals the capacity of an $s-t$ cut $\{X, \overline{X}\}$ of the network, i.e., there exists an $s-t$ cut $\{X, \overline{X}\}$ such that*

$$v(f) = C(\{X, \overline{X}\}).$$

In such case it must be true that

$$f_{ij} = c_{ij}, \quad if \quad (i,j) \in (X, \overline{X}),$$
$$f_{ji} = 0, \quad if \quad (j,i) \in (\overline{X}, X).$$

The general inverse maximum flow problem under the sum-type combining norms can be formulated as follows.

$$\begin{aligned} \min \ & \sum_{(i,j)\in A} \{w_{ij}H(c_{ij}, d_{ij}) + k_{ij} \cdot \max\{p_{ij} - d_{ij}, 0, d_{ij} - q_{ij}\}\} \\ \text{s.t. } & \text{Flow } f^0 \text{ is a maximum flow of } N(V, A, d); \\ & -l_{ij} \le d_{ij} - c_{ij} \le u_{ij}, \qquad \text{for each } (i,j) \in A. \end{aligned} \tag{2}$$

Note that in order to make the above problem and other inverse problems discussed in this paper well-posed, we always assume that there exists no directed $t - s$ path over which every arc (k, l) has a positive flow f^0_{kl}. In fact if this assumption does not hold, then no matter how to adjust the capacity vector c, in every $s - t$ cut $\{X, \overline{X}\}$, there would be a backward arc $(j, i) \in (\overline{X}, X)$ with $f^0_{ji} > 0$, and by Lemma 1, it is impossible to let f^0 become a maximum flow.

Lemma 2. *If problem (2) has a feasible solution, then there exists an optimal solution d^* such that $d^*_{ij} \le c_{ij}$ for all $(i,j) \in A$.*

Proof. Let d^* be an optimal solution of problem (2), then f^0 is the maximum flow of the network $N(V, A, d^*)$. From Lemma 1, we know that there is a minimum $s-t$ cut $\{X, \overline{X}\}$ of network $N(V, A, d^*)$ such that for each forward arc $(i,j) \in (X, \overline{X})$, $f^0_{ij} = d^*_{ij}$. Now if there exists an arc $(x, y) \in A$ with $d^*_{xy} > c_{xy}$, then of course $(x, y) \notin (X, \overline{X})$, for otherwise we would have $f^0_{xy} > c_{xy}$ which violates the feasibility of f^0. Define $\overline{d}$ as

$$\overline{d}_{ij} = \begin{cases} c_{ij}, & \text{if } (i,j) = (x,y), \\ d^*_{ij}, & \text{otherwise.} \end{cases}$$

Because $(x, y) \notin (X, \overline{X})$, in this $s - t$ cut $\{X, \overline{X}\}$, the capacity of $\overline{d}$ and the value of flow f^0 are still equal, which means that in the network $N(V, A, \overline{d})$, $\{X, \overline{X}\}$ is still a minimum $s - t$ cut and f^0 is still a maximum flow. From $-l_{ij} \le d^*_{ij} - c_{ij} \le u_{ij}$ for each $(i,j) \in A$ and the definition of $\overline{d}$, we know that $-l_{ij} \le \overline{d}_{ij} - c_{ij} \le u_{ij}$ for each $(i,j) \in A$. So $\overline{d}$ is a feasible solution of problem (2). However, we have

$$\begin{aligned} & \sum_{(i,j)\in A} \{w_{ij}H(c_{ij}, d^*_{ij}) + k_{ij} \cdot max\{p_{ij} - d^*_{ij}, 0, d^*_{ij} - q_{ij}\}\} \\ & - \sum_{(i,j)\in A} \{w_{ij}H(c_{ij}, \overline{d}_{ij}) + k_{ij} \cdot max\{p_{ij} - \overline{d}_{ij}, 0, \overline{d}_{ij} - q_{ij}\}\} \\ & = w_{xy} + k_{xy} \cdot max\{0, d^*_{xy} - q_{xy}\} \ge 0. \end{aligned}$$

If $w_{xy} + k_{xy} \cdot max\{0, d^*_{xy} - q_{xy}\} > 0$, then d^* cannot be an optimal solution of problem (2), a contradiction. Otherwise, $\overline{d}$ is another optimal solution of problem (2), but it satisfies $\overline{d}_{xy} = c_{xy}$. Hence, by repeating the above argument, we can conclude that there exists an optimal solution d^* of problem (2) such that $d^*_{ij} \le c_{ij}$ for all $(i,j) \in A$, i.e., $d^* \le c$. □

According to Lemma 2, no arc needs to increase its capacity in order to make f^0 form a maximum flow. Hence, the second group of constraints in problem (2) (and also in other inverse problems discussed in this paper) can be changed to

$$-l_{ij} \le d_{ij} - c_{ij} \le 0, \qquad \text{for each } (i,j) \in A.$$

That is to say, all bounds u_{ij} for increasing capacities are immaterial. So in this paper, without loss of generality, we only need to introduce l_{ij}, and assume they are all nonnegative and finite.

For the network $N(V, A, c)$ and the flow f^0, we can construct its residual network $N^{'}(V, A(c), c^{'})$ by the following algorithm which is introduced in [1].

Algorithm 1

Step 1. The node set is still V.

Step 2. If $(i, j) \in A$ and $f_{ij}^0 < c_{ij}$, then $(i, j) \in A(c)$ and $c_{ij}^{'} = c_{ij} - f_{ij}^0$.

Step 3. If $(i, j) \in A$ and $f_{ij}^0 > 0$, then $(j, i) \in A(c)$ and $c_{ji}^{'} = f_{ij}^0$.

We denote the arc set created by Steps 2 and 3 as $A(c)_1$ and $A(c)_2$, respectively.

And hence we have the following lemma.

Lemma 3. *Flow f^0 is a maximum flow of the network $N(V, A, c)$ if and only if its residual network $N^{'}(V, A(c), c^{'})$ has a maximum flow with value 0.*

In the following, for each arc set Γ we define $w^s(\Gamma) = \sum_{(i,j)\in\Gamma} w_{ij}$, $w^b(\Gamma) = \max_{(i,j)\in\Gamma} w_{ij}$ and use similar notation for other vector (here letters s and b stand for 'sum' and 'bottleneck', respectively).

Due to Lemma 2, problem (2) can be formulated as follows:

$$\begin{aligned} \min \ & \sum_{(i,j)\in A} \{w_{ij} H(c_{ij}, d_{ij}) + k_{ij} \cdot \max\{p_{ij} - d_{ij}, 0\}\} \\ \text{s.t. Flow } & f^0 \text{ is a maximum flow of } N(V, A, d); \\ & -l_{ij} \le d_{ij} - c_{ij} \le 0, \qquad \text{for each } (i, j) \in A. \end{aligned} \tag{3}$$

Lemma 4. *If problem (3) is feasible, then there exists an optimal solution d^* such that if $d_{ij}^* \neq c_{ij}$, then $d_{ij}^* = f_{ij}^0$ and hence $-l_{ij} \le f_{ij}^0 - c_{ij}$.*

Proof. If problem (3) is feasible, it must have an optimal solution. We first show that there exists an optimal solution d^* of problem (3) such that

$$d_{ij}^* = f_{ij}^0, \quad \text{if } d_{ij}^* \neq c_{ij}. \tag{4}$$

In fact if we have an optimal solution d which does not satisfy (4), i.e., if there exists an arc $(x, y) \in A$ such that $f_{xy}^0 < d_{xy} < c_{xy}$, and we can define d^* as

$$d_{ij}^* = \begin{cases} c_{ij}, & \text{if } (i, j) = (x, y), \\ d_{ij}, & \text{otherwise}, \end{cases}$$

and by an argument similar to the proof of Lemma 2, we can prove that d^* is again an optimal solution of (3), but now $d_{xy}^* = c_{xy}$ and hence the arc (x, y) no longer violates the requirement (4). In other words the number of arcs violating the requirement is reduced by 1. By repeating the process if necessary, finally we obtain an optimal solution d^* of problem (3) that meets (4).

Next, we show that the optimal solution d^* of problem (3) satisfies

$$-l_{ij} \le f_{ij}^0 - c_{ij}, \quad \text{if } \ d_{ij}^* \neq c_{ij}.$$

In fact, if there are some arcs (i,j) satisfy $-l_{ij} > f^0_{ij} - c_{ij}$, i.e., $f^0_{ij} < c_{ij} - l_{ij}$. Then, no matter how to adjust the capacity c_{ij}, we have $d^*_{ij} \geq c_{ij} - l_{ij} > f^0_{ij}$ due to the second constraint of the problem (3). But the above analysis shows that if $d^*_{ij} \neq c_{ij}$ then $d^*_{ij} = f^0_{ij}$, hence we need do nothing for those arcs. The lemma follows. □

Before going to present an algorithm to solve the problem (3) in strongly polynomial time, let us first explain the main motivations of the algorithm:

1. If the given flow f^0 is a maximum flow of the network $N(V,A,c)$, then we have nothing to do. Otherwise, we have $v(f^*) > 0$ due to the Lemma 3, where the f^* is a maximum flow of the residual network $N'(V,A(c),c')$ respect to $N(V,A,c)$ and f^0. Hence, we must to modify the capacity of the network $N(V,A,c)$ as little as possible such that the corresponding residual network $N'(V,A(c),c')$ has a maximum flow with value 0.
2. Denote $M = \{(i,j) \in A \mid -l_{ij} \leq f^0_{ij} - c_{ij} < 0\}$. Then we have nothing to do for the arcs $(i,j) \in A \setminus M$ due to Lemma 4. And if we modify an arc $(i,j) \in M$, we set $d^*_{ij} = f^0_{ij}$, hence the associate modification cost of the arc (i,j) is $\overline{w}_{ij} = w_{ij} + k_{ij} \cdot \max\{p_{ij} - f^0_{ij}, 0\}$.
3. If we investigate the effects of changing the capacities of arcs in the initial network $N(V,A,c)$ onto the residual network $N'(V,A(c),c')$, we observe that an existing arc can be deleted form $N'(V,A(c),c')$ since we decreas the capacity c_{ij} to f^0_{ij} for the arcs in M.

Combining the above analysis, we have the following result.

Theorem 1. *The problem (3) is equivalent to the following problem.*

$$\begin{aligned} &\min \overline{w}^s(X,\overline{X}) \\ s.t.\ &(X,\overline{X}) \subseteq A(c)_1; \\ &\{X,\overline{X}\} \text{ is an } s-t \text{ cut of } N^{'}(V,A(c),c^{'}); \\ &c_{ij} - l_{ij} \leq f^0_{ij}, \text{ for each } (i,j) \in (X,\overline{X}). \end{aligned} \tag{5}$$

Therefore, finding an optimal solution of problem (3) is equivalent to finding an optimal solution of problem (5). To solve problem (5) in strongly polynomial time, we further modify the residual network $N^{'}(V,A(c),c^{'})$ to $N^{''}(V,A(c),c^{''})$ in the following way: the node set and the arc set are unchanged; and the capacity of each arc is set as

$$c^{''}_{ij} = \begin{cases} \overline{w}_{ij}, & \text{if } (i,j) \in A(c)_1 \text{ and } c_{ij} - l_{ij} \leq f^0_{ij}, \\ (\frac{n^2}{4}+1)L, & \text{otherwise}, \end{cases} \tag{6}$$

where L is an upper bound for all $\overline{w}_{ij}$.

Theorem 2. *Let $\{X^*,\overline{X^*}\}$ be a minimum $s-t$ cut of the network $N^{''}(V,A(c),c^{''})$ with a capacity $c^{''s}(X^*,\overline{X^*})$.*

(1) If $c^{''s}(X^*,\overline{X^*}) \leq \frac{n^2}{4}L$, then $(X^*,\overline{X^*})$ must be the optimal solution of problem (5).

(2) If $c^{''s}(X^*,\overline{X^*}) > \frac{n^2}{4}L$, then problem (5) has no feasible solution.

Proof. (1) First, if $c^{''s}(X^*, \overline{X^*}) \leq \frac{n^2}{4}L$, then $c''_{ij} = \overline{w}_{ij}$ for all $(i,j) \in (X^*, \overline{X^*})$, i.e., for each $(i,j) \in (X^*, \overline{X^*})$, $(i,j) \in A(c)_1$ and $c_{ij} - l_{ij} \leq f^0_{ij}$. Second, as $N'(V, A(c), c')$ and $N''(V, A(c), c'')$ have the same node set and arc set, $(X^*, \overline{X^*})$ is also an $s-t$ cut of $N'(V, A(c), c')$. So, $(X^*, \overline{X^*})$ is a feasible solution of problem (5).

Moreover, it is easy to see that $\{X^*, \overline{X^*}\}$ is an optimal solution of problem (5). If not, suppose there exists an $s-t$ cut $\{X, \overline{X}\}$ which is feasible to problem (5), and $\overline{w}^s(X, \overline{X}) < \overline{w}^s(X^*, \overline{X^*})$. Then from (6), we have

$$c^{''s}(X, \overline{X}) = \sum_{(i,j)\in(X,\overline{X})} c''_{ij} = \overline{w}^s(X, \overline{X})$$

$$< \overline{w}^s(X^*, \overline{X^*}) = \sum_{(i,j)\in(X^*,\overline{X^*})} c''_{ij} = c^{''s}(X^*, \overline{X^*}),$$

which contradicts the fact that $\{X^*, \overline{X^*}\}$ is a minimum $s-t$ cut of $N''(V, A(c), c'')$.

(2) Suppose that $c^{''s}(X^*, \overline{X^*}) > \frac{n^2}{4}L$ but problem (5) has a feasible solution $\{X, \overline{X}\}$. From (6), we know that $c''_{ij} = w_{ij} \leq L$ for all $(i,j) \in (X, \overline{X})$. It implies that the capacity of $\{X, \overline{X}\}$ satisfies $c^{''s}(X, \overline{X}) \leq \frac{n^2}{4}L$ (as the cardinality $|(X, \overline{X})| \leq \frac{n^2}{4}$), which contradicts the fact that $\{X^*, \overline{X^*}\}$ is the minimum $s-t$ cut of the network $N''(V, A(c), c'')$. □

Now we are ready to give a full description of an algorithm to solve problem (3).

Algorithm 2

Step 1 For the network $N(V, A, c)$ and the given flow f^0, construct its residual network $N'(V, A(c), c')$, and then modify it to $N''(V, A(c), c'')$ according to formula (6).

Step 2 Find a minimum $s-t$ cut $\{X^*, \overline{X^*}\}$ of the network $N''(V, A(c), c'')$. If the capacity of the minimum cut satisfies $c^{''s}(X^*, \overline{X^*}) > \frac{n^2}{4}L$, then problem (3) has no feasible solution, stop. Otherwise, go to Step 3.

Step 3 Output an optimal solution d^* of problem (3) as

$$d^*_{ij} = \begin{cases} f^0_{ij}, & \text{if } (i,j) \in (X^*, \overline{X^*}), \\ c_{ij}, & \text{otherwise,} \end{cases}$$

and the associated optimal value $\overline{w}^s(X^*, \overline{X^*})$.

It is clear that Step 1 takes $O(m)$ time. Step 2 to find a minimum $s-t$ cut $\{X^*, \overline{X^*}\}$ of the network $N''(V, A(c), c'')$ takes $O(n^3)$ time [1]. Hence, Algorithm 2 runs in $O(m+n^3) = O(n^3)$ time in the worst-case, and it is a strongly polynomial algorithm.

3 Inverse Maximum Flow Problem under the Bottleneck-Type Combining Norms

The problem considered in this section is the inverse maximum flow problem under the bottleneck-type combining norms which can be formulated as follows:

$$\begin{aligned} &\min \max_{(i,j)\in A} \{w_{ij}H(c_{ij},d_{ij}) + k_{ij}\cdot\max\{p_{ij}-d_{ij},0\}\} \\ &\text{s.t. Flow } f^0 \text{ is a maximum flow of } N(V,A,d); \\ &\quad -l_{ij} \le d_{ij}-c_{ij} \le 0, \qquad \text{for each } (i,j)\in A. \end{aligned} \tag{7}$$

Note that problems (3) and (7) have the same feasible region, hence by a similar argument as in Lemma 2 we can obtain the following result.

Lemma 5. *If problem (7) is feasible, then there exists an optimal solution d^* such that if $d^*_{ij} \neq c_{ij}$, then $d^*_{ij} = f^0_{ij}$ and hence $c_{ij} - l_{ij} \le f^0_{ij}$.*

By an argument similar to the Section 2, we have the following result.

Theorem 3. *The problem (7) is equivalent to the following problem.*

$$\begin{aligned} &\min \overline{w}^b(X,\overline{X}) \\ &s.t.\ (X,\overline{X}) \subseteq A(c)_1; \\ &\quad \{X,\overline{X}\} \text{ is an } s-t \text{ cut of } N'(V,A(c),c'); \\ &\quad c_{ij}-l_{ij} \le f^0_{ij}, \text{ for each } (i,j)\in(X,\overline{X}), \end{aligned} \tag{8}$$

where $N'(V,A(c),c')$ is the residual network corresponding to $N(V,A,c)$ and f^0, $\overline{w}_{ij} = w_{ij} + k_{ij}\cdot\max\{p_{ij}-f^0_{ij},0\}$.

According to the above theorem, it suffices to find an optimal solution of problem (8). To solve problem (8) in strongly polynomial time, we again modify the residual network $N'(V,A(c),c')$ to $N''(V,A(c),\tau)$ by defining τ as follows:

$$\tau_{ij} = \begin{cases} \overline{w}_{ij}, \text{ if } (i,j)\in A(c)_1 \text{ and } c_{ij}-l_{ij} \le f^0_{ij}, \\ 2L, \text{ otherwise,} \end{cases} \tag{9}$$

where L bears the same meaning as in Section 2. Similar to Theorem 2, we can prove the following result.

Theorem 4. *If $\{X^*,\overline{X^*}\}$ is an $s-t$ cut of $N''(V,A(c),\tau)$ with the minimum bottleneck capacity, i.e.,*

$$\tau^b(X^*,\overline{X^*}) = \min\{\tau^b(X,\overline{X}) \mid \{X,\overline{X}\} \text{ is an } s-t \text{ cut of } N''(V,A(c),\tau)\}, \tag{10}$$

then we have:

(1). If $\tau^b(X^*,\overline{X^*}) \le L$, then $\{X^*,\overline{X^*}\}$ must be an optimal solution of problem (8).

(2). If $\tau^b(X^*,\overline{X^*}) > L$, then problem (8) has no feasible solution.

So far we have transformed problem (7) into the problem (8). Now we are ready to give a full description of an algorithm to solve problem (7).

Algorithm 3

Step 1 For the network $N(V,A,c)$ and the given flow f^0, construct its residual network $N'(V,A(c),c')$, and then modify it to $N''(V,A(c),\tau)$ according to formula (9).

Step 2 Find an $s-t$ cut $\{X^*, \overline{X^*}\}$ of the network $N^{''}(V, A(c), \tau)$ with the minimum bottleneck capacity. If $\tau^b(X^*, \overline{X^*}) > L$, then problem (7) has no feasible solution, stop. Otherwise, go to Step 3.

Step 3 Output an optimal solution d^* of problem (7) as

$$d_{ij}^* = \begin{cases} f_{ij}^0, & \text{if } (i,j) \in (X^*, \overline{X^*}), \\ c_{ij}, & \text{otherwise,} \end{cases}$$

and the associated optimal value $\overline{w}^b(X^*, \overline{X^*})$.

It is clear that Step 1 takes $O(m)$ time. Step 2 to find an $s-t$ cut $\{X^*, \overline{X^*}\}$ of the network $N^{''}(V, A(c), \tau)$ with the minimum bottleneck capacity takes $O(m + n \log n)$ time [8]. Hence, Algorithm 3 runs in $O(m + n \log n)$ time in the worst-case, and it is a strongly polynomial algorithm.

4 Concluding Remarks

In this paper we studied the inverse maximum flow problem under the combining norms. For sum-type and bottleneck-type objective functions, we transformed them into minimum-sum and minimum-bottleneck $s-t$ cut problems, respectively, and hence presented strongly polynomial algorithms to solve them.

As a future research topic, it will be meaningful to consider other inverse combinational optimization problems under combining norms. Studying computational complexity results and proposing optimal/approximation algorithms are promising.

Acknowledgments. The author wish to thank the anonymous referees for their helpful comments.

References

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms and Applications. Prentice-Hall, New York (1993)
2. Burton, D., Toint, P.: On an instance of the inverse shortest paths problem. Mathematical Programming 53, 45–61 (1992)
3. He, Y., Zhang, B., Yao, E.: Wighted inverse minimum spanning tree problems under Hamming distance. Journal of Combinatorial Optimization 9, 91–100 (2005)
4. Heuberger, C.: Inverse Optimization: A survey on problems, methods, and results. Journal of Combinatorial Optimization 8, 329–361 (2004)
5. Liu, L., He, Y.: Inverse minimum spanning tree problem and reverse shortest-path problem with discrete values. Progress in Natural Science 16, 649–655 (2006)
6. Liu, L., Yao, E.: A weighted inverse minimum cut problem under the bottleneck type Hamming distance. Asia-Pacific Journal of Operational Research 24, 725–736 (2007)
7. Liu, L., Zhang, J.: Inverse maximum flow problems under the weighted Hamming distance. Journal of Combinatorial Optimization 12, 395–408 (2006)

8. Schrijver, A.: Combinatorial Optimization, Polyhedra and Efficiency. Springer, Berlin (2003)
9. Yang, C., Zhang, J., Ma, Z.: Inverse maximum flow and minimum cut problems. Optimization 40, 147–170 (1997)
10. Zhang, J., Xu, S., Ma, Z.: An algorithm for inverse minimum spanning tree problem. Optimization Methods and Software 8, 69–84 (1997)
11. Zhang, B., Zhang, J., Qi, L.: The shortest path improvement problems under Hamming distance. Journal of Combinatorial Optimization 12, 351–361 (2006)

The Edge-Recoloring Cost of Paths and Cycles in Edge-Colored Graphs and Digraphs*

Carlos A. Martinhon[1] and Luérbio Faria[2]

[1] Fluminense Federal University, Department of Computer Science, Niterói, RJ, Brazil
[2] Estadual Univ. of Rio de Janeiro, Inst. of Computation, Rio de Janeiro-RJ, Brazil
mart@dcc.ic.uff.br, luerbio@cos.ufrj.br

Abstract. In this paper we introduce a number of problems regarding edge-color modifications in edge-colored graphs and digraphs. Consider a property π, a c-edge-colored graph G^c (resp., digraph D^c) not satisfying π, and an edge-recoloring cost matrix $R = [r_{ij}]_{c\times c}$ where $r_{ij} \geq 0$ denotes the cost of changing color i of edge e to color j. Basically, in this kind of problem the idea is to change the colors of one or more edges of G^c (resp., oriented edges in D^c) in order to construct a new c'-edge-colored $G^{c'}_{new}$ with $c' \leq c$ (resp., $D^{c'}_{new}$) such that the total edge-recoloring cost is minimized and property π is satisfied. Here, we are especially concerned with properly edge-colored and monochromatic paths, trails and cycles in graphs and digraphs.

Keywords: Edge-colored graphs, properly edge-colored paths, trails and cycles, edge-recoloring cost.

1 Introduction, Notation and Terminology

In the last few years a great number of applications have been modelled as problems in edge-colored graphs and digraphs with important applications in molecular biology [11,12], large communication networks [5], social sciences [4], resolution of strategic conflicts [13], among others.

Given a graph $G = (V, E)$, a walk ρ from s to t in G (called *s-t walk*) is a sequence $\rho = (v_0, e_0, v_1, e_1, \ldots, e_k, v_{k+1})$ where $v_0 = s$, $v_{k+1} = t$ and $e_i = v_iv_{i+1} \in E$ for $i = 0, \ldots, k$. A trail from s to t in G (called *s-t trail*) is a walk ρ from s to t where $e_i \neq e_j$ for $i \neq j$. Analogously, a path from s to t is a trail where $v_i \neq v_j$ for $0 \leq i < j \leq k$ and $1 \leq i < j \leq k + 1$. If ρ is a path (resp., trail) with $v_0 = v_{k+1}$ then ρ defines a *cycle* (resp., *closed trail*). We define $V(\rho) = (v_0, v_1, \ldots, v_{k+1})$ and $E(\rho) = (e_0, e_1, \ldots, e_k)$. The *length* of a path, trail or walk is the number of its edges.

Let $I_c = \{1, 2, \ldots, c\}$ be a set of given colors, with $c \geq 2$. Here, G^c denotes a simple graph whose edges are colored by colors of I_c and with no parallel edges linking the same pair of vertices. The vertex and edge sets of G^c are denoted

* This work was partially supported by FAPERJ (Projects E-26/110.552/2010 and E-26/103.054/2011) and CNPq/Brazil.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 231–240, 2013.

by $V(G^c)$ and $E(G^c)$, respectively, where $|V(G^c)| = n$ and $|E(G^c)| = m$. For a given color i, $E^i(G^c)$ denotes the set of edges of G^c colored by i. The degree of x in G^c is $d_{G^c}(x) = \sum_{i \in I_c} |N^i_{G^c}(x)|$ or just $d(x)$ when no confusion arises. An edge (resp., arc or oriented edge) between two vertices x and y is denoted by xy (resp., $\boldsymbol{xy}$), its color by $c(xy)$ (resp., $c(\boldsymbol{xy})$) and its weight by $w(xy)$ (resp., $w(\boldsymbol{xy})$), if any.

In this work, we consider a c-edge-colored graph G^c (resp., digraph D^c), a property π and an edge recoloring matrix $R = [r_{ij}]_{c \times c}$ where $r_{ij} \geq 0$ denotes the cost of changing color i of edge $e \in E^i(G^c)$ to color j. If property π does not hold in G^c (resp., D^c), we consider the problem of changing one or more colors of edges in G^c (resp., D^c) such that property π is satisfied and the total edge-recoloring cost is minimized. In this kind of problem, colors can be associated to different communications links, frequencies or channels and must be changed whenever some desired structure with a given color pattern is no longer verified. The objective is to construct or re-establish a particular structure (possible denied by an attacker) with the minimum edge-recoloring cost. Obviously, according to the situation, the objective can also be destroy some existing unwanted (or malicious) structures in the graph.

Here, we are specially concerned with *properly edge-colored* (or just PEC for short) and monochromatic structures such as paths, trails and cycles. A subgraph of G^c containing at least two edges is said to be a PEC *subgraph* if any two adjacent edges differ in color. A PEC *path* (resp., PEC *trail*) is a path (resp., trail) such that any two successive edges have different colors. Given two vertices $s, t \in V(G^c)$, we call PEC s-t path (resp., PEC s-t trail) a path (resp., trail) that begins at s and finishes at t. In addition, observe that edges in a PEC trail need not to form a PEC subgraph since we can have adjacent but not successive edges with the same color. A PEC path or trail in G^c is said to be *closed* if its endpoints coincide and its first and last edges differ in color. In the oriented case, we denote properly arc-colored paths and trails, respectively, by PAC paths and PAC trails for short. A *monochromatic* structure in G^c contains all its edges colored with the same color. Finally, we say that matrix R above satisfies the *recoloring triangle inequality*, if and only if, for all colors i, j, k of I_c, we have $r_{ij} \leq r_{ik} + r_{kj}$, meaning that color i of edge e can be changed at most one time. Unless specified otherwise matrix R is always symmetric.

In Section 2, we deal with the determination of monochromatic paths, perfect matching and spanning trees with the minimum edge-recoloring cost. In Section 3 we consider PEC paths (resp., PAC trails) in edge-colored graphs (resp., arc-colored digraphs). Finally, in Section 4, we consider the problem of destroying undesirable structures such as PEC cycles and PEC closed trails in G^c. Related problems are also proposed.

2 Construction of Monochromatic Structures in G^c

Given G^c with $c \geq 2$ and an edge-recoloring matrix R, we deal with the problem of determining the minimum edge-recoloring cost necessary to construct a new graph $G^{c'}_{new}$ (for $c' \leq c$) containing some particular monochromatic structures

such as s-t paths, perfect matchings and spanning trees. We show that all these problems can be trivially solved in polynomial time by converting them into classical algorithms for paths, perfect matchings and spanning trees (see [1]). We conclude the section with the problem of finding the minimum edge-recoloring cost necessary to determine two edge-disjoint monochromatic s-t paths with different colors.

Theorem 1. *The problem of finding the minimum edge-recoloring cost of monochromatic s-t paths, perfect matchings and spanning trees can be solved in polynomial time.*

Proof: Here we only deal with the monochromatic s-t path problem, the other cases are analogous. Consider G^c with $c \geq 2$, $s, t \in V(G^c)$ and an edge-recoloring matrix R as above. Basically, the idea is to construct a sequence of weighted non-colored graphs G'_k for $k = 1, \ldots, |I_c|$ and compute a shortest path between s and t (respectively, minimum perfect matching and minimum spanning tree). To do that, we perform the following steps:

(i) Set $Cost(G^{c'}_{new}) \leftarrow +\infty$; {the value of the minimum edge-recoloring cost in $G^{c'}_{new}$}
(ii) For each color $k = 1, .., |I_c|$ do
- (ii.1) Construct G'_k with $V(G'_k) = V(G^c)$ and non-colored $E(G'_k) = \{e : e \in E(G^c)\}$;
- (ii.2) For each $i \in I_c$ assign non-negative weights $w_e \leftarrow r_{ik}$ for all edges $e \in E^i(G^c)$;
- (ii.3) Compute a shortest s-t path in G'_k, say P_k, with cost $Cost(P_k)$;
- (ii.4) If $Cost(P_k) < Cost(G^{c'}_{new})$ then set $k' \leftarrow k$ and $Cost(G^{c'}_{new}) \leftarrow Cost(P_k)$;

(iii) To construct $G^{c'}_{new}$ containing a monochromatic s-t path, change the colors of all edges (other than k') to color k' in the shortest s-t path $P_{k'}$ in G^c;
(iv) Return $G^{c'}_{new}$ with the monochromatic s-t path $P_{k'}$ and cost $Cost(G^{c'}_{new})$.

Note that since $|I_c| \leq |E(G^c)| = m$, the procedure above is obviously polynomial with total complexity time equal to $O(m.n^2)$, where $O(n^2)$ is the complexity of the shortest s-t path problem. □

Corollary 1. *Give $L \leq m$, the problem of finding the minimum edge-recoloring cost of a monochromatic s-t path of length at most L can be solved in polynomial time.*

Proof: To prove our result we deal with the Shortest Restricted Path problem, or SRP problem for short. In the SRP problem we are given a digraph $D = (V, E)$ with specified vertices $s', t' \in V$, a positive integer $L' > 0$, weights $w_{ij} \in Z^+$ and lengths $\ell_{ij} \in Z^+$ associated to each arc $\boldsymbol{ij} \in A$. The objetive is to determine the minimum shortest path (with respect to w) between s' and t' with length at most L'. The SRP problem is **NP**-hard (see [6]), however it can be solved in

polynomial time whenever all weights or lengths are polynomially bounded. In Hassin [10], the author describe a dynamic programming procedure for the SRP problem with pseudo-polynomial complexity $O(|E|.L')$.

To prove that, it suffices to construct G_k in the procedure of Theorem 1 above for $k = 1, .., |I_c|$ and set $\ell_{ij} = 1$ for every edge ij of $E(G_k)$. Then, we substitute Step $(ii.3)$ by the dynamic programming procedure detailed in [10] for the SRP problem. Since $L' = L \leq m$, this dynamic programming procedure is obviously polynomial. □

Now, we prove that the determination of the minimum edge-recoloring cost to obtain 2 edge-disjoint monochromatic s-t paths with different colors is **NP**-hard. The authors in [8] prove that determine whether an arbitrary G^c contains two vertex-disjoint monochromatic s-t paths with different colors is **NP**-complete (they do not consider edge-recoloring costs). Nevertheless, note that determining two edge-disjoint monochromatic s-t paths can be trivially solved in polynomial time. As a consequence, we can establish the following result (in the decision version) combining edge-recoloring costs and monochromatic edge-disjoint s-t paths:

Theorem 2. *Consider G^c with $c \geq 2$, an edge-recoloring matrix R of order c, two vertices $s, t \in V(G^c)$ and an integer $W \geq 0$. Then, the problem of determining 2 edge-disjoint monochromatic s-t paths with different colors and edge-recoloring cost less or equal than W is* **NP***-complete.*

3 Construction of PEC Paths, Trails and Cycles

3.1 The Non-Oriented Case

As discussed in [2], given G^c and $s, t \in V(G^c)$, the problem of determining whether or not G^c contains a PEC path between s and t can be solved in polynomial time. Thus, if G^c contains no PEC s-t paths we can establish the following result:

Theorem 3. *The problem of determining the minimum edge-recoloring cost necessary to construct a* PEC *s-t path can be solved in polynomial time.*

The following problem was left open:

Problem 1: Consider G^c, $s, t \in V(G^c)$, an edge-recoloring matrix R and positive constant $L \leq m$. What is the complexity of determining the minimum edge-recoloring cost necessary to construct a PEC s-t path of length at most L?

3.2 The Oriented Case

Now, consider a digraph D^c with vertices $s, t \in V(D^c)$ and an arc-recoloring matrix R of order $c \geq 2$. In Gourvès et. al. [9], the authors show that maximize the number of PAC s-t trails in D^c can be done in polynomial time. Surprisingly, they prove that the determination of one PAC s-t path is **NP**-complete even for planar arc-colored digraphs containing no PAC circuits. Thus, if D^c contains no

s-t trails, we consider the problem of determining the minimum arc-recoloring cost necessary to construct $D^{c'}_{new}$ (with $c' \leq c$) containing a directed PAC s-t trail. Formally:

Theorem 4. *The problem of finding the minimum arc-recoloring cost necessary to construct a* PAC *s-t trail can be solved in polynomial time.*

Proof: Initially, *w.l.o.g.* suppose that vertices s and t, respectively, contains no outgoing and incoming arcs in D^c. Basically, the idea is to construct an weighted non-colored digraph D such that the solution of the minimum cost flow problem over D indicates the best arc-color modifications to construct $D^{c'}_{new}$ containing a directed PAC s-t trail with the minimum arc-recoloring cost.

To do that, we define gadgets D_x for each $x \in W = V(D^c) \setminus \{s, t\}$ with the following vertex- and arc-sets:

$$V(D_x) = \left(\bigcup_{i \in I_c} \{x_i, x'_i\} \right) \quad and \quad A(D_x) = \bigcup_{\{i \in I_c\}} \bigcup_{\{j \in I_c\}} \{\boldsymbol{x_i x'_j}\}. \tag{1}$$

Now, we define weights and capacities of the arcs $\boldsymbol{x_i x'_j} \in A(D_x)$ as in the sequel. We set $w(\boldsymbol{x_i x'_j}) = +\infty$ for $i = j$, and set $w(\boldsymbol{x_i x'_j}) = 0$ for $i \neq j$. All capacities of the arcs in $A(D_x)$ are settled to 1. In addition, we change vertices s and t of D^c, respectively, by s' and t' in the digraph D.

To conclude our construction of D, for each arc $\boldsymbol{xy} \in A(W)$ with color $c(\boldsymbol{xy}) = i$, we introduce $|I_c|$ arcs $\boldsymbol{x'_j y_j}$ in D for $j \in I_c$ and set $w(\boldsymbol{x'_j y_j}) = r_{ij}$ (the cost of changing color i to color j). Note that, if $i = j$ then $w(\boldsymbol{x'_j y_j}) = 0$. Finally, for every arc $\boldsymbol{sx}$ (resp., $\boldsymbol{yt}$) of D^c with color $c(\boldsymbol{sx}) = i$ (resp., $c(\boldsymbol{yt}) = i$) we introduce arcs $\boldsymbol{s' x_j}$ (resp., $\boldsymbol{y'_j t'}$) and set $w(\boldsymbol{s' x_j}) = r_{ij}$ (resp., $w(\boldsymbol{y'_j t'}) = r_{ij}$) for every $j \in I_c$. The capacities of all remaining arcs of D are settled to 1. See the example of Figures 1.(a) and 1.(b).

If a sequence ρ denotes a solution of the minimum cost flow problem over D, we can easily determine all arc-color modifications with the minimum arc-recoloring cost necessary to construct D^c_{new}. To see that, for every arc $\boldsymbol{x'_i x_j} \in A(\rho)$ of D with $w(\boldsymbol{x'_i x_j}) = r_{ij} > 0$, we change the color of the associated arc $\boldsymbol{xy}$ of D^c from i to j (since $i \neq j$), otherwise, the color of $\boldsymbol{xy}$ remains the same. Finally, note that sequence ρ^* obtained in this way defines a PAC s-t trail in D^c_{new}. To see that, note at every gadget G_x of D, we have $w(\boldsymbol{x_i x'_i}) = +\infty$ for $i \in I_c$. Then, the solution ρ of D contains no arcs $\boldsymbol{x_i x'_i}$ avoiding the presence of consecutive arcs colored alike in the associated ρ^* of D^c_{new}. □

Corollary 2. *Given $L > 0$, the problem of finding the minimum arc-recoloring cost necessary to construct a* PAC *s-t trail of length at most $L \leq m$ can be solved in polynomial time.*

Proof: To prove that, it suffices to construct digraph D as above and set $\ell_{xy} = 1$ for each arc $\boldsymbol{xy}$ of $A(D)$. Then, we apply the dynamic programming procedure as described in Hassin [10] to find a shortest s-t path in D with total length at most $L' = 2L - 1$. □

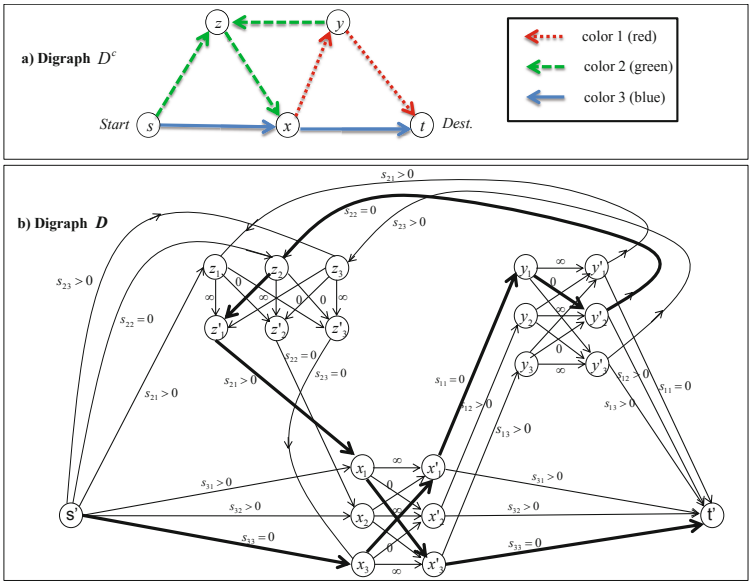


Fig. 1. In (a), we have an arc-color digraph D^c with 3 colors containing no PAC s-t trails. In (b), after solving the minimum cost flow problem over D, suppose that we send one unit of flow along the path $\rho = (s', x_3, x'_1, y_1, y'_2, z_2, z'_1, x_1, x'_3, t')$ with $cost(\rho) = w(\boldsymbol{z'_1 x_1}) = r_{21} > 0$ (denoted in bold). Thus, after changing the color of arc $\boldsymbol{zx}$ from green (color 2) to color red (color 1) in D^c we obtain a new arc-colored digraph D^c_{new} and 2 PAC s-t trails $\rho^*_1 = (s, x, y, z, x, t)$ and $\rho^*_2 = (s, z, x, t)$, with the minimum arc-color edge-recoloring cost equal to $r_{21} > 0$.

4 Destruction of PEC Cycles and PEC Closed Trails in G^c

The authors in Yeo [15] and Abouelaoualim *et. al.* [2] present a polynomial time characterization to determine, respectively, c-edge-colored graphs containing no PEC cycles and PEC closed trails. Hence, a polynomial time algorithm can be constructed to guarantee the existence of PEC cycles (resp., closed trails) in G^c, if any. Several interesting problems can be solved in polynomial time when restricted to these particular classes of graphs. For instance, as discussed in [2], if G^c contains no PEC cycles (resp., closed trails) and $s, t \in V(G^c)$, the problem of finding a longest PEC s-t path (resp., s-t trail) in G^c can be solved in polynomial time. Further, as proved in Gourvès *et. al.* [8], if Ψ is a subset of $V(G^c) \setminus \{s, t\}$ the problem of finding a PEC s-t path (resp., s-t trail) visiting all vertices of Ψ can be solved in polynomial time, provided that one exists (this problem is NP-complete over general c-edge-colored graphs).

Therefore, PEC cycles or PEC closed trails can be viewed as undesirable structures and a related question is to determine the minimum edge-recoloring cost necessary to destroy them. Unfortunately, this problem is **NP**-hard. Formally, we have the following associated decision problem:

Theorem 5. *Consider G^c with $c \geq 2$, an asymmetric edge-recoloring matrix R and an integer $W \in Z^+$. Then, the problem of finding a new c-edge-colored graph G^c_{new} containing no* PEC *cycles (resp., closed trails) and edge-recoloring cost less or equal than W is* **NP***-complete.*

Proof: This problem obviously belong to **NP**. Given a subset of edges and colors to be modified in G^c, we can polynomially verify, through the Yeo's algorithm (resp., bridge's algorithm), if the resulting new edge-colored graph G^c_{new} contains no PEC cycles (resp., closed trails).

To construct our reduction we deal with the E3SAT problem. An instance $\mathcal{B} = \Lambda_{k=1}^{m} C_k$ in the *Conjunctive Normal Form* - CNF of the E3SAT consists of a set of n variables $\mathcal{X} = \{x_1, \ldots, x_n\}$ and m clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$ with exactly three literals per clause [6]. In addition, for $\ell \in \{1, 2, 3\}$, we say that x_i is the ℓ-th variable of C_k, if and only if, x_i and exactly $\ell - 1$ other variables $x_{i'}$ with $i' < i$ appear in C_k. Thus, given $\mathcal{B}$ we show how to polynomially determine a 2-edge-colored graph G^c, a matrix S of order 2 and an integer $k \geq 0$ such that the edge-recoloring cost necessary to construct G^c_{new} with no PEC cycles is less or equal than $W = n$, whenever $\mathcal{B}$ is satisfiable for some truth assignment of $\mathcal{B}$. The destruction of PEC closed trails is analogous and will be considered later.

Our construction proceed is as follows. For each variable x_i of $\mathcal{X}$ we introduce 4 vertices $u_i^1, u_i^2, v_i^1, v_i^2$ with edges $u_i^1 u_i^2$ and $v_i^1 v_i^2$ colored *red*, and edges $u_i^1 v_i^1$ and $u_i^2 v_i^2$ colored *blue*. Let us denote by G_i, this gadget associated to x_i. Both *red* edges $u_i^1 u_i^2$ and $v_i^1 v_i^2$ of G_i, will be associated to literals x_i and $\bar{x}_i$, respectively. Now, for each clause C_k and pair of variables x_i, x_j of C_k occurring, respectively, in the *ℓ-th* and *h-th* positions of C_k with $h \equiv (\ell mod\, 3) + 1$ for $\ell = 1, 2, 3$, we introduce a *blue* edge $e_k(\ell, h)$ in the following manner:

(i) if literal x_i appears in the *ℓ-th* position of C_k we connect vertex u_i^2 with vertex u_j^1 (if literal x_j appears in the *h-th* position of C_k) or v_j^1 (if literal $\bar{x}_j$ appears in the *h-th* position of C_k).

(ii) if literal $\bar{x}_i$ appears in the *ℓ-th* position of C_k we connect vertex v_i^2 with vertex u_j^1 (if literal x_j appears in the *h-th* position of C_k) or v_j^1 (if literal $\bar{x}_j$ appears in the *h-th* position of C_k).

We repeat this construction for each $k = 1, .., m$. After that, multiple pairs of blue edges connecting the same pair of vertices, if any, will be substituted by one single edge, colored *blue* in G^c. Now, if we assume that *red* is the color 1, and *blue* is the color 2 in I_c, to construct the edge-recoloring cost matrix $R = [r_{ij}]_{2\times 2}$ we set $r_{ij} = 0$ for $i = j$, $r_{12} = 1$ and $r_{21} = n + 1$. Finally, we set $W = |\mathcal{X}| = n$. A complete example of our construction is presented in Figure 2. Note that each PEC cycle in G^c containing red edges of any combination of three gadgets will be associated to a clause C_k of $\mathcal{C}$ for some $k \in \{1, .., m\}$. Further, if all of these PEC cycles are destroyed (by changing the color of some of its red edges) all the remaining PEC cycles will be destroyed (see Figure 2).

Therefore, if we have a satisfiable truth assignment for $\mathcal{B}$, we can construct a new c-edge-colored graph G^c_{new} containing no PEC cycles and such that the total edge-recoloring cost is no greater than n. To do that, for each index $i \in \{1, .., n\}$, we change the color of edge $u_i^1 u_i^2$ (resp., $v_i^1 v_i^2$) from *red* to *blue*, whenever literal

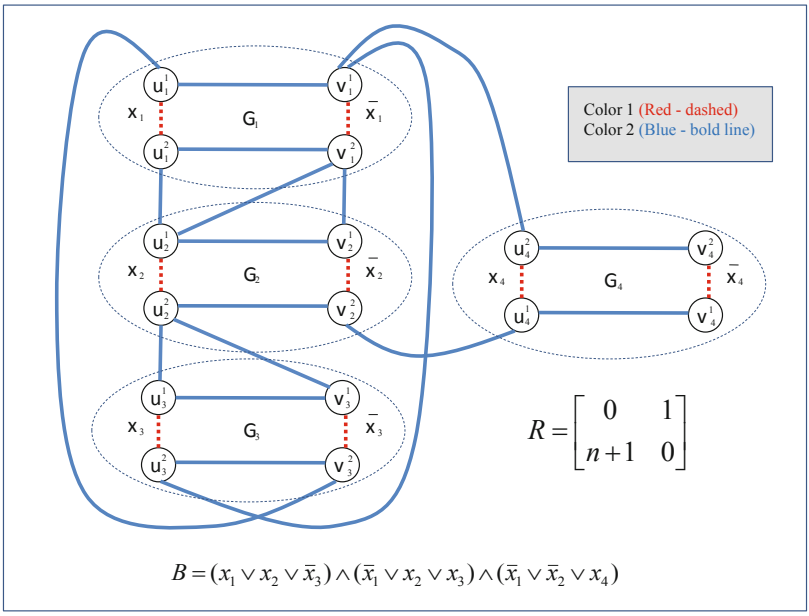


Fig. 2. The destruction of PEC cycles is **NP**-complete. In the example, clauses $C_1 = (x_1 \vee x_2 \vee \bar{x}_3)$, $C_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$ and $C_3 = (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$ are associated, respectively, with PEC cycles $C_1 = (u_1^1, u_1^2, u_2^1, u_2^2, v_3^1, v_3^2, u_1^1)$, $C_2 = (v_1^1, v_1^2, u_2^1, u_2^2, u_3^1, u_3^2, v_1^1)$ and $C_3 = (v_1^1, v_1^2, v_2^1, v_2^2, u_4^1, u_4^2, v_1^1)$, all of them containing red edges of 3 different gadgets.

x_i is *true* (resp., *false*) in the assignment. Note that if x_i is *true* (resp., *false*) for $i = 1, .., n$, all PEC cycles of G^c containing edges $u_i^1 u_i^2$ (resp., $v_i^1 v_i^2$) will be destroyed in G^c_{new}. In addition, since n edge colors are modified from color 1 to color 2 with $r_{12} = 1$, the total color edge-recoloring cost will be equal to n. Reciprocally, if all PEC cycles of G^c are destroyed with an edge recoloring cost not exceeding n, we can easily obtain a satisfiable truth assignment for $\mathcal{B}$. Initially, note that since $r_{2,1} = n+1$, only the color of *red* edges must be changed. Further, exactly one *red* edge of each gadget G_i must be modified, otherwise, after n modifications we should have at least one non-destroyed PEC cycle in G^c_{new} associated to G_i for some $i \in \{1, .., n\}$. Therefore, if the color of *red* edge $u_i^1 u_i^2$ (resp., $v_i^1 v_i^2$) is modified, we set literal $x_i = true$ (resp., $x_i = false$) in the assignment. Note that, each time a PEC cycle associated to a clause C_j of $\mathcal{B}$ is destroyed, at least one variable of C_j will be true in the assignment. Thus, if all PEC cycles are destroyed we have a satisfiable truth assignment for $\mathcal{B}$, which concludes this part of the proof.

Finally, to guarantee the destruction of PEC closed trails in G^c it suffices to substitute each *blue* edge $e_k(\ell, h) = xy$ for k, ℓ, h as above; by a sequence of vertices x, w_1, w_2, w_3, y with edges xw_1, w_2w_3, w_1y colored *blue* and edges

w_1w_2, w_3w_1 colored *red*. Note that this transformation is not applied over the *blue* edges of gadgets G_i for $i = 1, .., n$. Therefore, using the same arguments as above we prove that the PEC closed trail case is also **NP**-complete. □

The authors in [2], show how to polynomially determine, if an arbitrary G^c contains no PEC cycles and almost PEC cycles through a vertex x (we say that a cycle is an almost PEC cycle through vertex x if the first and last edges incident to x have the same color). To illustrate the importance of this class of graphs, they consider the problem of maximizing the number of vertex disjoint PEC paths between s and t. They prove that this problem can be solved in polynomial time if G^c contains no PEC cycles and almost PEC cycles through s or t. However, it becomes **NP**-hard for c-edge-colored graphs containing no PEC cycles (note that almost PEC cycles through s or t are allowed in this case). Further results regarding s-t paths restricted to this particular class of graphs can be found in [2]. Unfortunately, we have the following result consequence of Theorem 5 above:

Corollary 3. *Consider G^c with $c \geq 2$, an asymmetric matrix R of order c, an integer $W \in Z^+$ and a vertex $x \in V(G^c)$. Then, the problem of finding a new c-edge-colored graph G^c_{new} with no (almost)* PEC *cycles through x and edge-recoloring cost less or equal than W is* **NP***-Complete.*

Proof: To prove that we add a new vertex x_k and change the blue edges $e_k(3, 1)$ for $k = 1, .., m$ (connecting the last and first gadgets of clause k). In other words, all edges $e_k(3, 1)$ are substituted by 2 blue edges incident at x_k. Now, after contracting all vertices x_k by a new vertex $\bar{x}$ our proof follows as in Theorem 5. □

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory. Algorithms and Applications. Prentice-Hall (1993)
2. Abouelaoualim, A., Das, K.C., Faria, L., Manoussakis, Y., Martinhon, C.A., Saad, R.: Paths and trails in edge-colored graphs. Theoretical Computer Science 409(3), 497–510 (2008)
3. Berman, P., Karpinski, M., Scott, A.D.: Hardness of Short Symmetric Instances of MAX-3SAT. Elect. Colloquium on Comput. Complexity (49), 1433–8092 (2003)
4. Easley, D., Kleinberg, J.: Networks, crowds and markets: Reasoning about a highly connected world, Cambridge (2010)
5. Gamvros, I.: Satellite network design, optimization and management, PhD thesis, University of Maryland (2006)
6. Garey, M.R., Johnson, D.: Computers and intractability: a guide to the theory of np-completeness. W. H. Freeman and Company, New York (1979)
7. Gerards, A.M.H.: Matching, *Network Models.* In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (eds.) Handbooks in Operations Research and Management Science, vol. 7. North-Holland, Amsterdam (1995)
8. Gourvès, L., Lyra, A., Martinhon, C., Monnot, J.: On paths, trails and closed trails in edge-colored graphs. Discrete Mathematics and Theoretical Computer Science 14(2), 57–74 (2012)

9. Gourvès, L., Lyra, A., Martinhon, C., Monnot, J.: Complexity of trails, paths and circuits in arc-colored digraphs. To Appear in the Disc. Appl. Math (2012)
10. Hassin, R.: Approximation schemes for the restricted shortest path problem. Math. of Oper. Res. 17(1), 36–42 (1992)
11. Pevzner, P.A.: DNA physical mapping and properly edge-colored eurelian cycles in colored graphs. Algorithmica 13, 77–105 (1995)
12. Pevzner, P.A.: Computational molecular biology: an algorithmic approach. MIT Press (2000)
13. Xu, H., Li, K., Kilgour, M., Hipel, K.: A matrix-based approach to searching colored paths in a weighted colored multidigraph. Appl. Math. and Comp. 215, 353–366 (2009)
14. Sands, B., Sauer, N., Woodrow, R.E.: On monochromatic paths in edge-coloured digraphs. J. Comb. Theory (B) 33, 271–275 (1982)
15. Yeo, A.: A note on alternating cycles in edge-coloured Graphs. Journal of Comb. Theory, Series B 69, 222–225 (1997)

A Cost-Efficient Scheduling Algorithm for Traffic Grooming

Xianrong Liu, Wenhong Tian*, Minxian Xu, and Qin Xiong

School of Computer Science and Software Engineering, University of Electronic Science and Technology of China, Chengdu, P.R. China
tian_wenhong@ustec.edu.cn

Abstract. In this paper, we consider a fundamental traffic grooming problem in optical line-system: the number of total links with lengths and the max number of wavelengths (capacity) of each fiber are given, also a set of demands (jobs) and their routes are fixed so that the load of each link is known, the problem is to construct a set of fiber intervals so that the total fiber length is minimized (called Fiber Lengths Minimization problem or FLM for abbreviation). It is known that FLM problem is NP-complete in general. In this paper, we propose a 2-approximation algorithm, Longest Link interval First (LLF), which is better than existing best known bound.

Keywords: Traffic Grooming, Fiber Lengths Minimization problem, Minimizing Total Fiber Length, Longest Link interval First (LLF).

1 Introduction

In optical network design, decomposing the network into a set of optical line systems is one way of avoiding the expensive O-E-O (optical to electrical to optical) conversion [13]. In this way, system becomes transparent, only demands between different linesystems need O-E-O conversion; also routing is not necessary in this case since wavelength assignment problem can be solved separately in each linesystem. All-optical networks have been extensively studied in recent years, especially for the core networks. A logical path formed by a signal traveling from its source to its destination using a unique wavelength is termed a lightpath. If the nodes have no conversion capability, then the requirement that the same wavelength must be used on all the links along the selected route is known as the wavelength continuity constraint and makes networking significantly different from conventional circuit switched networks. Our work assumes that the nodes are not capable of wavelength conversion.

The network usually supports traffic that is at rates that are lower than the full wavelength capacity, and therefore the network operator has to be able to put together (groom) low-capacity connections into the high capacity lightpaths. The network operator often has to groom low-capacity demands into high capacity

* Corresponding author.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 241–249, 2013.

fibers to save cost (energy and equipment costs etc.). This can be viewed as assigning colors to the lightpaths so that at most g of them can share the same fiber, where $g \geq 1$ is the capacity of a single fiber. This grooming problem is one of very important issues in optimizing costs for optical networks. With MOADMs (mesh optical add/drop multiplexer), there is polynomial time solution for this problem [13].

Without MOADMs, the linesystem can be treated as a collection of fibers each of which occupies an interval of the line, the problem is called packing intervals in intervals, which is NP-complete (see a proof given in [13]; in this case, each demand must be assigned not only a wavelength but also a fiber which covers the intended interval; deploying fiber satisfying all demands but minimizing total length (MinLength) of fibers is NP-hard when total number of wavelength (μ) is larger than 1. In this paper, the case without MOADMs and wavelength conversion is considered. The book [3] provides many research results about scheduling algorithms that may be applied in job allocations. The paper [13] discusses wavelength assignment and generalized interval graph coloring and provides NP-complete proof for the problem. [8] reviews recent research into the energy-efficiency in optical networks. [7] summarizes recent technologies for reducing the power consumption of optical access networks. [9] [5] discusses the regenerator placement and routing in translucent optical networks. [4] provides approximating solution for traffic grooming with respect to ADMs and OADMs. [2] provides a ($logM$) and ($log\mu$) -approximation algorithms for minimizing total number of fibers where M is the number of links in this system. [6] proposes a general 4-approximation algorithm for minimizing total number of OADMs. [10] discusses the online version of this scheduling problem. In this paper, 2-approximation algorithm is proposed.

2 Problem Formulation

The problem can be formally stated as follows: an optical line-system has n links $e_1, e_2, \ldots, e_n$, with link e_i carrying fibers and each fiber can carry g wavelengths, the length of link e_i is L_i. Represent a demand by $[i, j]$ for $i \leq j$ if it requires links $e_i, \ldots, e_j$; the set of demands (jobs) will be denoted by D. The load $l_i = l(e_i)$ on link e_i is the minimum number of fibers required to carry all the demands on link e_i, where d_i is the number of demands on link e_i. Consider demands D are given, together with link lengths, the objective is to construct a set F of fiber intervals of minimum total length which can satisfy D, it is called Fiber Lengths Minimization problem (FLM problem for abbreviation).

Theorem 1. *The lower bound for FLM problem is the sum of the minimum number of fibers used on each link multiplies the length of each link.*

Proof: For a given set of jobs J and demands D, we can find the minimum number of fibers needed for each link, denoted as $l_i, l_2, \ldots, l_k$,

$$l_i = l(e_i) = \lceil \frac{d_i}{g} \rceil \tag{1}$$

for total k links under consideration, where l_i is the minimum number of fibers needed for link e_i. Then ideally, the min length of all fibers is the sum of (the minimum number of fibers used in each link) multiplies (the length of each link (denoted as L_i)), i. e. :

$$MinLength(OPT) = \sum_{i}^{n} l_i L_i \tag{2}$$

Example 1: As shown in Fig.1. , there are four requests j_1 to j_4 and three

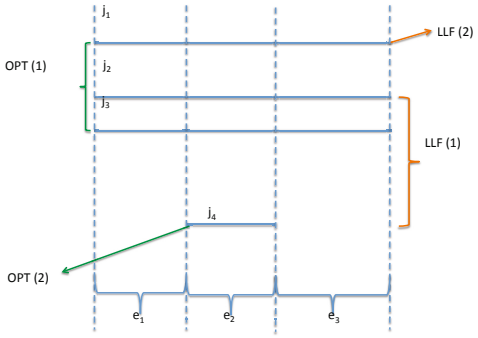


Fig. 1. An example of MinLength problem

links e_1, e_2, e_3. j_1, j_2 and j_3 pass through link e_1, e_2 and e_3; j_4 passes through link e_2. Each fiber has wavelengths (capacity) $g = 3$. Therefore, the lower bound of total number of fibers needed on link e_1 to e_3 is 1, 2, 1 respectively; and the total length of all fibers is $(L_1 + 2L_2 + L_3 = l_1L_1 + l_2L_2 + l_3L_3)$, i. e., the sum of (the minimum number of fibers used in each link) multiplies (the length of each link (denoted as L_i)).

Observation 1: The lower bound for FLM problem is to allocate exactly $\lceil \frac{d}{g} \rceil$ number of fibers to each link, where d_i is the fiber length on link i.

Remark 1: The lower bound is not easy to achieve. One way to achieve this is to apply First Fit Decreasing (FFD) algorithm in [6] to sort all requests in non-increasing order of their spans, and allocate the subset of longest span jobs first. By sorting all requests in non-increasing order of their spans and allocate the subset of longest span jobs first, the long span jobs will not be distributed to too many other fibers, so that the total fiber length may be minimized. However, FFD may not work well in some cases. It is shown that FFD has approximation ratio 4 in the worst case [6].

In the following, we consider that the nodes have no wavelength conversion capability. The paper [6] showed that it is NP-hard to approximate our problem

already in special case where all jobs have the same span and can be allocated on one fixed link, by a simple reduction from the subset sum problem. To see the hardness of the FLM problem, THEOREM 2 is given as follows:

Theorem 2. *FLM problem is NP-complete problem in general case.*

We sketch a simpler proof than [10] as follows:
Proof: In the following, we show that the well-known NP-complete problem, K-PARTITION problem can be reducing to FLM problem in polynomial time. K-PARTITION problem is well-known NP-complete (see [3] and reference therein): for a given arrangement S of positive numbers and an integer K. Partition S into K ranges so as to sums of all the ranges are close to each other. K-PARTITION problem can be reduced to our FLM problem as follows. For a set of jobs J, each has capacity demand and span constraints (set as positive numbers), partitioning J by their capacities into K ranges, is the same to allocate K ranges of jobs with capacity constraint g (i.e. the sum of each range is at most g). On the other hand, if there is a solution to K-PARTITION for a given set of intervals, there exists a schedule for the given set of numbers. Since K-PARTITION is NP-hard in the strong sense, our problem is also NP-hard. In this way, we have found that that our FLM problem is NP-complete problem.

Definition 1. *Approximation ratio: an offline deterministic algorithm is said to be C-approximation for the objective if it obtains results in a polynomial time at most C times that of an optimal solution.*

Since the general FLM problem is NP-complete, in the following, we propose an efficient approximation algorithm.

3 The Approximation Algorithm: Longest Link Interval First

In this section, a 2-approximation algorithm called Longest Link interval First (LLF) is introduced. The LLF algorithm is described in Algorithm 3.1. The LLF algorithm allocates the requests from the longest link interval to the shortest interval. Each job is scheduled to the first fiber which can fit. This algorithm has computational complexity $O(N \max(M, logN))$ where N is the number of jobs and M is the number of fibers needed on any link. Because The LLF algorithm firstly sorts all jobs (requests) in non-decreasing order of their start points (line 1), this takes $O(NlogN)$ time. The load of each link is represented by min number of fibers needed (line 2-4). Then the algorithm finds a fiber for a request needs $O(M)$ steps where M is the min number of fibers need on any link (line 5-12), N jobs altogether need $O(MN)$ steps. Therefore, the entire algorithm takes $O(N\ max(M, logN))$ time where normally $N > M$.

Example 2: As shown in Fig. 1 where each fiber can carry max g=3 wavelengths. Without loss of generality, assuming that link length $L_2 > L_1 = L_3$. According

Input: A job (demand) instance $J = \{j_1, j_2, \ldots, j_n\}$, and the max capacity g of a fiber (g is the grooming papameter)
Output: The allocated jobs and total length of all fibers
Sort all jobs in non-decreasing order of their start-points (s_i for job i), such that $s_1 \leq s_2 ... \leq s_n$, set f=1 **forall the** links under consideration **do**
  represent load of link e_i by the min number of fibers needed, denoted as l_i (take integral value by ceiling function).
end
forall the jobs under consideration **do**
  Find the longest continuous link interval with same load first, denoted as $[z_1, z_2]$; If two link interval have same length, consider larger load first
  forall the jobs ended or started in $[z_1, z_2]$ **do**
    always consider the longest job when other parameters are the same;
    allocate to the first fiber which can fit, use a new fiber and set f=f+1 if needed
  end
  remove allocated jobs, update load of each link
end
Count load of all links and total length of all fibers.

Algorithm 3.1: Longest Load First Algorithm

to LLF algorithm, j_4 is allocated firstly to the first fiber on link e_2 since the longest link interval is on it, j_2 and j_3 are also allocated to the first fiber on link e_2 since j_1, j_2 and j_3 have the same start-point and length. j_1 is then allocated to another fiber on link e_2 since g=3 on any fiber. Notice that j_1, j_2 and j_3 can be allocated in any order in this case.

Theorem 3. *The approximation ratio of our proposed Longest link interval First(LLF) algorithm for FLM problem has an upper bound 2.*

Proof: We provide a proof by induction. Consider there are n requests and a fiber can carry g wavelengths.

1. Since one fiber can host at most g requests, let firstly consider n=$g + 1$, we have $LLF(J) \leq 2OPT$ in this case. The adversary is that these $g + 1$ jobs have different start-points, end-points, shorter jobs are contained by the longer ones, and are sorted in non-decreasing order of their start-points as shown in Fig. 3(b) where f is set as 2 in this case. The total fiber length of optimal solution is dominated by the length of the longest job with span T_1, (g+1)-th job with span T_{g+1} (assuming it is the shortest span length but has link length longer that other links). LLF treats most-load links first when two load spans have same length, its total fiber length is dominated by the 2-nd longest job with span T_2, and the longest job with span T_1 (one job left for a single fiber). Therefore we have:

$$\frac{LLF(J)}{OPT(J)} = \frac{T_1 + T_2}{T_1 + T_{g+1}} = \frac{1 + \frac{T_2}{T_1}}{1 + \frac{T_{g+1}}{T_1}} \tag{3}$$

Equ. (3) will have upper bound 2 when T_1=T_2 and other span lengths are negligible comparing to T_1; for other cases, LLF(J) equals to OPT(J).

2. Assuming that $LLF(J) \leq 2\ OPT(J)$ holds for n=k under clique, one-sided clique, container and other cases. And there are total f fibers used. Let us denote the optimal solution and LLF solution as OPT_k and LLF_k respectively, we have:

$$LLF_k \leq 2OPT_k \tag{4}$$

3. Next, we consider n=k+1. For this case, there are following situations after sorting all $k+1$ jobs in increasing order of their start-points:

   (a) The total number of fibers needed is still f, i.e., the $(k+1)$-th job can be allocated to one of f existing fibers. There are following two sub conditions:
   i). The $(k+1)$-th job can be allocated to one of f existing fibers and the total fiber length of all fibers will not change, i.e., LLF_k=LLF_{k+1} and $OPT_k = OPT_{k+1}$. In this case, obviously, $LLF_{k+1} \leq 2OPT_{k+1}$ holds.
   ii). Assuming that the allocation of $(k+1)$-th job will increase the total fiber length of LLF and OPT by l_{k+1} for the upper bound. i. e., LLF_{k+1}=$LLF_k + l_{k+1}$, OPT_{k+1}=OPT_k+l_{k+1}, and $l_{k+1} \leq len(j_{k+1})$. (As for other scenarios, such as the $(k+1)$-th job only increases the total fiber length of LLF (i.e., $(k+1)$-th job is contained by some longer jobs) or only increases the total fiber length of OPT, one can easily check that $LLF_{k+1} \leq 2OPT_{k+1}$ holds). We then have:

$$LLF_k + t_{k+1} \leq 2OPT_k + t_{k+1} \leq 2OPT_k + 2t_{k+1} \leq 2OPT_{k+1} \tag{5}$$

   (b) The total number of fibers needed will increase by 1, i.e., $(f+1)$ fibers are needed. This means that the $(k+1)$-th job intersects with all existing jobs and cannot be hosted by any existing fiber. We consider the following three typical hard sub-conditions (other scenarios are trivial and easy to show so that we omitted the proofs for them) :
   i). One-sided clique: in this case, all job intervals form a one-sided clique, either started or ended at the same node as shown in Fig.2, assuming link e_1 has longest length. In this case, optimal solution is to allocate longest group of jobs to a fiber, the second longest group of jobs to another fiber, and so on. The total fiber length of optimal solution is dominated by the span length of the longest job with span T_1, $(fg+1)$-th job with span T_{fg+1} (the shortest one). Let us denote total length of other fibers as T_O in optimal solution. LLF treats most-load links first when two load span have same lengths, its total fiber length is dominated by the 2-nd longest job with span T_2, and the longest job with span T_1 (one job left for a single fiber), denote total length of other fibers as T_H. therefore:

$$\frac{LLF_{k+1}}{OPT_{k+1}} = \frac{T_1 + T_2 + T_H}{T_1 + T_{g+1} + T_O} = \frac{1 + \frac{T_2 + T_H}{T_1}}{1 + \frac{T_{g+1} + T_O}{T_1}} \tag{6}$$

Equ. (6) will have upper bound 2 when T_1=T_2 and other span lengths are negligible comparing to T_1; for other cases, LLF_{k+1} equals to OPT_{k+1}.

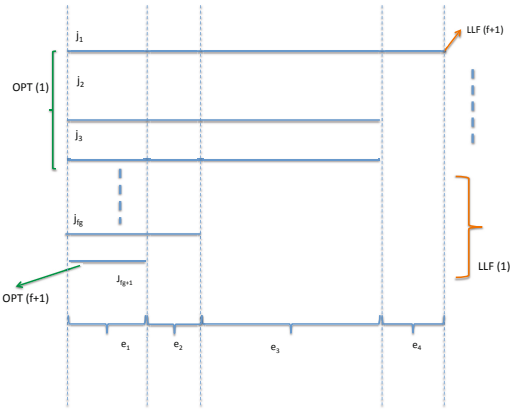


Fig. 2. One-sided clique case for LLF algorithm

ii). Clique case: Let us consider all jobs started and ended at different time as shown in Fig. 3 (a). The adversary is that two or more jobs with longest spans are spreading left and right across the point where all job intervals intersect, assuming the center link e_x (the cross point for all jobs) has longest length. Optimal solution will allocate others to one or more fiber and allocate j_{g+1} to a separate fiber with total fiber length of $(T_{g+1}+T_g+T_O)$; LLF algorithm will allocate j_{g+1} and j_g to one fiber, so on and the shortest one left for a single fiber, let set it as j_1. therefore:

$$\frac{LLF_{k+1}}{OPT_{k+1}} = \frac{T_{g+1}+T_g+T_H+T_1}{T_g+T_{g+1}+T_O} = \frac{1+\frac{T_H+T_1}{T_g+T_{g+1}}}{1+\frac{T_O}{T_g+T_{g+1}}} \tag{7}$$

Equ. (7) will have upper bound 2 when T_1+T_H=T_g+T_{g+1} and other span lengths are negligible comparing to T_g+T_{g+1}; for other cases, LLF_{k+1} equals to OPT_{k+1}. A similar 2-approximation algorithm by consider span distance is also provided in [6] for this case.

iii). The container case: The adversary is shown in Fig. 3 (b), i.e., shorter interval jobs are contained in longer interval jobs and assuming link e_x is longer than other links, this is one of the worst cases for LLF algorithm. Let us set these $(k+1)$ jobs have lengths T_1, T_2, ..., T_k, T_{k+1} in non-increasing order. The $(k+1)$-th job is the longest jobs for LLF, so

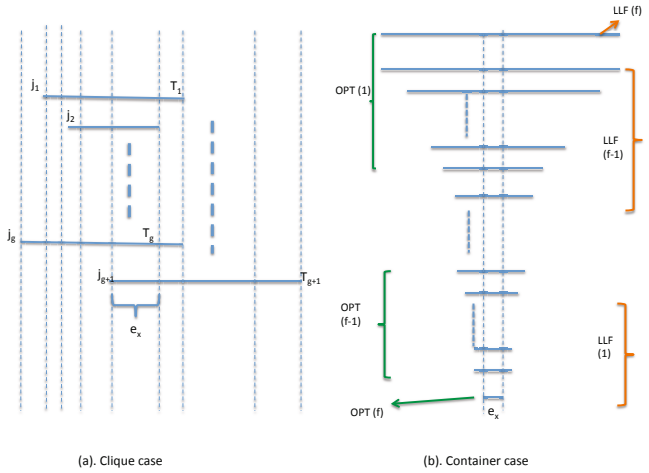


Fig. 3. Clique and Container Case for LLF algorithm

that $LLF_{k+1}=(T_{k+1}+LLF_{k-1})+T_1=T_{k+1}+LLF_k$. As for the optimal solution, one can allocates the longest job first, so $OPT_{k+1}=OPT_k+T_{k+1}$. Therefore $LLF_{k+1}=LLF_k+T_{k+1}\leq 2OPT_k+T_{k+1}\leq 2OPT_k+2T_{k+1}=2OPT_{k+1}$, this means $LLF_{k+1}\leq 2OPT_{k+1}$.

By combining the above analyses, we have proved Theorem 3.

4 Conclusion

In this paper, an efficient traffic-grooming algorithm, LLF, for minimizing total fiber length is proposed. Both theoretical lower bound and approximation are discussed. The proposed algorithm can help network designer to save the deployment cost, management cost and energy etc. We are still looking for near-optimal solution for this problem. There are a few more open research issues for the problem: including finding better near-optimal solution and providing theoretical proofs for the approximation algorithms; extending to other network topologies, like in [1][2][6]; considering stochastic demands such as in [11] [12]; and considering other optimization objectives. With the above-mentioned extensions and other related issues, it is possible to develop comprehensive cost-efficient methods for traffic grooming in optical networks.

References

1. Andrews, M., Zhang, L.: Bounds on fiber minimization in optical networks with fixed fiber capacity. In: Proceedings of IEEE INFOCOM 2005, Miami, FL (March 2005)

2. Andrews, M., Zhang, L.: Complexity of wavelength assignment in optical network optimization. IEEE/ACM Transactions on Networking 17(2), 646–657 (2009)
3. Brucker, P.: Scheduling Algorithms, 5th edn. Springer, Heidelberg (2007)
4. Flammini, M., Monaco, G., Moscardelli, L., Shalom, M., Zaks, S.: Approximating the traffic grooming problem with respect to adms and oadms. In: 14th International European Conference on Parallel and Distributed Computing (EuroPar), Las Palmas de Gran Canaria, Spain, August 26-29 (2008)
5. Flammini, M., Monaco, G., Moscardelli, L., Shalom, M., Zaks, S.: Optimizing Regenerator Cost in Traffic Groomings. Technical Report (July 2011)
6. Flammini, M., Monaco, G., Moscardelli, L., Shachnai, H., Shalom, M., Tamir, T., Zaks, S.: Minimizing Total Busy Time in Parallel Scheduling with Application to Optical Networks. Theoretical Computer Science 411(40-42), 3553–3562 (2010)
7. Kani, J., Shimazu, S., Yoshimoto, N., Hadama, H.: Energy-Efficient Optical Access Network Technologies. In: The Proceedings of OFC 2011 (2011)
8. Kilper, D.: Energy-Efficient Networks. In: The Proceedings of OFC 2011 (2011)
9. Pachnicke, S., Paschendaand, T., Krummrich, P.M.: Physical Impairment Based Regenerator Placement and Routing in Translucent Optical Networks. In: The Proceedings of OFC-NFOEC 2008 (2008)
10. Shalom, M., Voloshin, A., Wong, P.W.H., Yung, F.C.C., Zaks, S.: Online Optimization of Busy Time on Parallel Machines. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 448–460. Springer, Heidelberg (2012)
11. Tian, W.: A Dynamic Modeling And Dimensioning Approach For Optical Networks. In: 3rd International Conference on Broadband Communications, Networks and Systems, BROADNETS 2006, October 1-5 (2006)
12. Tian, W.H.: Modeling approaches and provisioning algorithms for dynamic bandwidth adjustment in multi-media networks. In: 2009 HONET, International Symposium on High Capacity Optical Networks and Enabling Technologies (2009)
13. Winkler, P., Zhang, L.: Wavelength assignment and generalized interval graph coloring. In: SODA, pp. 830-831 (2003)

Strategies of Groups Evacuation from a Convex Region in the Plane

Yinfeng Xu[1,2] and Lan Qin[1,2,⋆]

[1] School of Management, Xi'an Jiaotong University, Xi'an, 710049, China
[2] State Key Lab for Manufacturing Systems Engineering, Xi'an, 710049, China
yfxu@mail.xjtu.edu.cn, qinlan@stu.xjtu.edu.cn

Abstract. How to evacuate from affected area when an emergency occurs? How to escape efficiently from the affected area when the evacuees are divided into multiple groups with complete information sharing with each other? This paper studies the evacuation strategies of this problem from a convex region in the plane, and analyzes this problem in two scenarios: general plane and plane in grid network. In these two scenarios, we design evacuation strategies and analyze the evacuate ratio of strategies, respectively. In some cases, we show that the given strategies are optimal. Furthermore, we analyze the performance of strategies by comparing them in different situations.

Keywords: Evacuation strategy, Groups evacuation, Grid network.

1 Introduction

In recent years, emergencies happen frequently, such as a fire in urban city, traffic accidents. Due to this, evacuation problems get more and more concern. Especially, people concern about how to evacuate efficiently from affected area in the event of emergency? The affected area could be regarded as a convex region, boundary information of the area is usually unavailable to affected people. The evacuees in the area may be divided into several groups to escape.

Previous research related to evacuation mainly focus on two problems: searching problem and evacuation problem. A number of papers in searching problem address the strategy in a plane or a graph of different variations. Deng[1] gave a competitive algorithm in the problem of learning an unknown environment. Burgard[2] considered the problem of exploring an unknown environment by a team of robots which is similar to multiple groups that we will consider in this paper. Fleischer[3] discussed searching a goal in an unknown environment from unknown position in different graphs. Batta[4] studied the problem of how to select optimal path when transporting dangerous goods. Panaite[5] designed an exploration algorithm which uses a robot to construct a complete map under unknown environment. Papadimitriou[6] made use of worst-case analysis to study

⋆ Corresponding author: School of Management, Xi'an Jiaotong University, Xianning West Road, Xi'an, 710049, China.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 250–260, 2013.

the shortest path problem on the graph. Searching problems are generally studied in the plane or on a graph. The goals of these problems are to search the boundary in a plane and to find all the edges or vertexes in a graph. However, in evacuation scenario, evacuees only need to get out of the region from an affected point in it. There is no need to explore the whole plane or graph.

Another attractive research area is the evacuation problem. Chen[7] used methods of system simulation to compare the evacuation efficiency of three different networks, which include the grid network we would also consider in this paper. Hamacher[8] described the difference between macroscopic and microscopic evacuation models and illustrated value of practical application of evacuation study which is equally applicable in our research. Lu[9] and Shekhar[10] studied the shortest path algorithm of evacuation with consideration of capacity constraints and the increasing number of people in time and space. These studies mainly focus on details of evacuation such as flow and other constraints to analyze the strategy under complete information without considering the actual situation, in which some information in evacuation may be unavailable.

Combining the two problems above, this paper considers evacuees only need to get to any point in the boundary of convex region in evacuation scenario. What's more, the evacuees don't know the boundary information of the region.

Such searching problems in unknown environment generally estimated by the performance ratio in the past, see for example the survey by Berman[11]. The performance of a search strategy is usually measured by competitive ratio, which is the ratio of search length and the shortest path length in an unknown environment. Thus, Fleischer[3] further present a similar ratio in the fixed graph, which is called search ratio. As mentioned above, we consider the evacuate ratio in this paper with the cost of strategy as time spent instead of path lengths [2]. In reality, the fast the evacuees run, the more safety. The time is more reliable to describe performance of evacuation than path length. The different between them is that time spent considers the situation of staying at a point for a while. It is a pity that this different don't show up in this paper, but this is important for strategies design in future.The time spent we studied is period from beginning of the evacuation to the moment all the evacuees are successfully evacuated from the region.

As we known, path length from one point to another is the shortest length which is traversed along the path in the graph between them. The shortest path in this paper is the path from starting point to the point whose path length is the shortest from point to any point on the boundary of affected region. Furthermore, the goal of this paper is to minimize the evacuate ratio which is quotient of time spent in the strategy over the time spent in the shortest path.

The structure of this paper is as follows. Section 2 describes the groups evacuation problem, and gives some assumptions. Section 3 analyzes the lower bound of this problem. Section 4 studies the evacuation problems of multiple groups in general plane. Section 5 analyzes evacuation problems when the evacuees are in one group, two groups and four groups in grid network, respectively.

Section 6 shows the performance of the strategies by comparing them in different situations. Section 7 concludes main results of the paper.

2 Preliminary

Suppose the affected area is a convex polygon M in a plane. Evacuees at point O in M have no knowledge of the boundary and the location in M. The goal of the evacuees is to evacuate from M as soon as possible. The evacuation strategies are proposed on two scenarios: general plane and plane with grid network. Multiple groups of evacuees can communicate with each other during the evacuation. The evacuees in different groups can share on-time information all the time. The evacuees successfully evacuate from convex region if they have reached the boundary of M. The evacuation time is the cost of strategy which is period from beginning of evacuation to the moment all the evacuees successfully evacuate from the convex region. The goal is to minimize the evacuate ratio of the strategy, which is the ratio of evacuation time in the strategy without boundary information to that by optimal with boundary information. The basic assumptions of this paper are as follows:

1. The evacuees don't know the boundary information of M, which includes the boundary $e = \{e_1, e_2, \cdots, e_j\}$ and their location in the M.

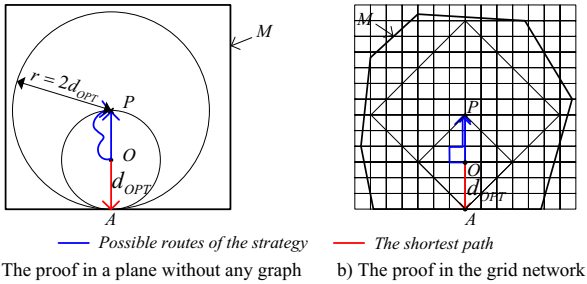
2. The evacuees move at unit speed from point O in the M to boundary of M. In grid network, the point O is a node in M.

3. When we study in grid network, assume the network consists of several grid units with edge length of 1. Evacuees travel along the edges of network and can not stay on the edge but the node of the network.

3 Lower Bound

In this section, we try to analyze the lower bound of the groups evacuation problem in the plane without the boundary information. Suppose the number of groups is k and multiple groups have full communication with each other.

Theorem 1. *In general plane, when boundary information is unknown, the evacuate ratio of any groups strategy for k groups is no less than 3.*



[!b]

Fig. 1. The proof of lower bound

Proof. Without loss of generality, suppose OA is the shortest path from point O to the boundary of M with intersection point A. The length of OA is d_{OPT} and the corresponding evacuation time is t_{OPT}.

For any k groups of strategy with full communication among them, let d_i denote the distance the group i has traversed from the point O. If the maximum distance satisfies $d_{\max} = \max\{d_1, d_2, \cdots, d_k\} < d_{OPT}$, they cannot reach the boundary of M. There exists a situation in which one of the groups has reached a point P first, where the path length from the point O to P is equal to d_{OPT}. Thus, we can construct a convex region as shown in Fig.1 a) above to make the point A in the shortest path opposite to the point P. And the path length from the point P to any point on the boundary of M is no less than $2d_{OPT}$. In the meantime, when the shortest path has been reached by an group, the length from the point O to A is at least d_{OPT}. Therefore, the cost of strategy without information is at least 3 times as the shortest path. The Theorem follows. □

For the scenario in grid network, we analyze the lower bound with similar assumptions as that in general plane for four groups and get the corollary as follows.

Corollary 1. *For any strategy without boundary information in grid network, the evacuate ratio of four groups is no less than 3.*

Proof. Similarly as the theorem 1, we can construct M as shown in Fig.1 b). Therefore, the evacuate ratio of four groups in grid network is no less than 3. □

4 Scenario 1: General Plane

For the situation of general plane, evacuees are divided into $k(k \geq 3)$ groups. Multiple groups of evacuees can completely share information with each other. We design an evacuation strategy as follows (as shown in Fig.2):

Equally Divided Exploration Strategy (EDES)

Step1: Divide the evacuees on O into k groups, i.e. $\{G_1, G_2, \cdots, G_k\}$.

Step2: Make k radials $\{r_1, r_2, \cdots, r_k\}$ from point O, as shown in Fig.2. For r_i to r_{i+1}, the angle between them is $\frac{2\pi}{k}$. The k groups of evacuees escape along k radials at time $t = 0$, respectively.

Step3: At time $t = t_i$, G_i is the first arrival at a boundary of M. Other evacuation groups that do not arrive at any boundary of M come back to the starting point O and walk along the path of G_i until they get out of M.

Definition 1. *Let C_e denote a circle with radius R' in M. Let the circle C_f with radius $R = \max R'$ be the largest inner circle of M. Fix the C_f's center in point O, the circle obtained is called O-center circle of M, denote as C_g with radius r.*

Theorem 2. *In general plane, the evacuate ratio of EDES with $k(k \geq 3)$ groups is no more than $\frac{3}{\cos(\pi/k)}$.*

Proof. According to the definition of circle C_g, we know convex region M has at least one intersection point with C_g. Without loss of generality, suppose P is one of the intersection points between r_i and r_{i+1}. As shown in Fig.3, tangent line intersects C_g, r_i and r_{i+1} at points P, H and Q, respectively. Obviously, path length d_i of any group G_i is no more than $\min\{OQ, OH\}$, and $d_i \leq \max\min\{OQ, OH\} = \frac{d_{OPT}}{\cos(\pi/k)}$, $t_{EDES} \leq 3t_i = 3d_i \leq \frac{3d_{OPT}}{\cos(\pi/k)}$. Hence, the evacuate ratio of EDES is $c = \frac{t_{EDES}}{t_{OPT}} \leq \frac{3}{\cos(\pi/k)}$.

The Theorem follows. □

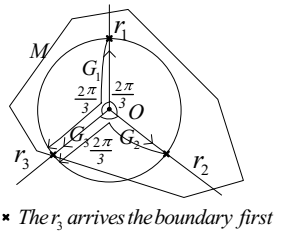


Fig. 2. The EDES when $k = 3$

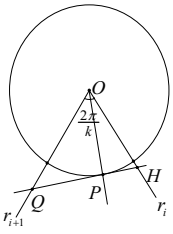


Fig. 3. The proof of theorem 2

The performance of EDES becomes more effective when k increasing. Thus, $\lim_{k\to+\infty}[3/\cos(\pi/k)] = 3$, this strategy is close to be optimal when groups is numerous. We can conclude that the number of groups which evacuees are divided into should be as many as possible.

5 Scenario 2: Plane with Grid Network

Most urban cities have the structure of road network like a grid one as shown in Fig.4, just as Zhang[12] described. In this section, we study the evacuation strategy in a plane with grid network.

Fig. 1. Part of the traffic network of Barcelona, Spain and Xi'an, Shannxi, China

For the sake of simplicity, with point O as origin to establish a rectangular coordinate system coincides with grid network. The shortest distance from intersections of coordinate axis and M to point O is a. Assume $P(x, y)$ denote coordinates of a point in grid network.

5.1 Analysis of Strategy with Boundary Information

In this part, we consider the evacuation problem with complete information. The goal of this problem is to find the shortest path. The cost of strategy is still the time spent of the strategy.

Definition 2. *For any points $P(x_1, y_1)$ and $P(x_2, y_2)$ in gird network, $L = |x_1 - x_2| + |y_1 - y_2|$ is called evacuation path length.*

Strategy with boundary information: all the evacuees walk along the shortest path OA from point O until they have reached region boundary.

Theorem 3. *The strategy with boundary information is optimal, and corresponding optimal evacuation time is $t_{OPT} = \lceil a \rceil$.*

Proof. Make a square $AEFG$ as shown in Fig.5. The function expressions of four edges of $AEFG$ respectively are: $x + y = a$, $x - y = a$, $-x + y = a$, $-x - y = a$. Hence, in closed region $AEFG$, path length from point O to any point inside of the closed region $AEFG$ satisfies $L^{in}_{AEFG} \leq a$, yet the path length from O to any point outside of $AEFG$ satisfies $L^{out}_{AEFG} > a$.

From the definition of OA, the square $AEFG$ has to intersect with M at point A. It can be easily obtained that other points on boundary of M are not inside of $AEFG$ from the properties of convex. Therefore, for any point $P(x, y)$ on boundary of M, $\min\limits_{P(x,y)\in e} \{L_{O,P(x,y)}\} = L_{O,A} = a$. Because of the assumptions in this paper, we obtain $t_{OPT} = \lceil L_{O,A} \rceil = \lceil a \rceil$.

The Theorem follows. □

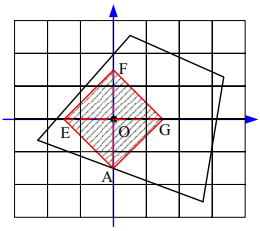


Fig. 5. The proof of Theorem 3

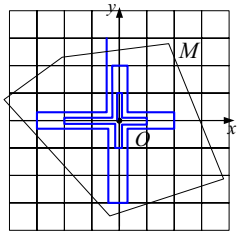


Fig. 6. The ORS when $l = 1$

5.2 Analysis of Strategies without Boundary Information

In reality, there are several different situations when emergency occurs. One is that all the evacuees converge to a same path to evacuate from affected area. The

other is that the evacuees are divided into several groups, and choose different routes to escape from affected area with complete information sharing among them. According to actual situation, we study three situations including the evacuees as one group, two groups and four groups, respectively.

5.2.1 Case 1: One Group

When disaster occurs, evacuees take a same path to escape from convex region. In this situation, we design a kind of One-group Reposition Strategy (as shown in Fig.6). The strategy description and analysis are as follows. In the strategy, l denotes the length of step unit, which can be any positive integer chosen by evacuees.

One-Group Reposition Strategy (ORS)

Step1: $t = 0$, $l, i, j \in N*$ and $l > 0$, $q = 1$, $i = 1$, $j = 0$. G_1 is located at point O. Let $D = \{D_1, D_2, D_3, D_4\} = \{+\overrightarrow{y}, +\overrightarrow{x}, -\overrightarrow{y}, -\overrightarrow{x}\}$ denotes directions in network.

Step2: G_1 walk towards D_q with length of $i \cdot l$. If they reach the boundary of M, stop; Otherwise, $j = j + 1$, retrace back to point O. If $q < 4$, $q = q + 1$;if $q = 4$, let $q = 1$.

Step3: If $j < 3$, go to Step2; if $j = 3$, let $i = i + 1$, $j = 0$, go to Step 2.

Lemma 1. *In grid network,* $\lceil a \rceil < \sqrt{2}R + 1$.

Proof. According to Theorem 3, square $AEFG$ is included in convex region. From definition of C_g, we obtain C_g intersects M at least one point S. Make the tangle line of C_g through the point S, say tl_s. If the tl_s intersects with axes at only one point, it satisfies $r = \min\{a, b, c, d\} = a$. If tl_s intersects with axes at two points, we say S_1, S_2. From property of convex, we obtain $a \leq \min\{OS_1, OS_2\} \leq \sqrt{2}r$. Hence, for any point of intersection satisfies it in the same way. Thus $\lceil a \rceil < a + 1 \leq \sqrt{2}r + 1 \leq \sqrt{2}R + 1$.

The Lemma follows. □

Theorem 4. *The evacuate ratio of ORS is less than* $3\frac{\sqrt{2}R+1}{l} + 6l - 2$.

Proof. Without loss of generality, let $\lceil a \rceil = (n-1) \cdot l + \lceil b \rceil$ where $\lceil b \rceil, n \in N*$ and $1 \leq \lceil b \rceil \leq l$. For any $(n-1) \cdot l < \lceil a \rceil \leq n \cdot l$, evacuation time of this strategy is $t_{ORS} \in [\lceil a \rceil + 3l \cdot n^2 - 3l \cdot n, \lceil a \rceil + 3l \cdot n^2 + 3l \cdot n]$, thus $c = \frac{t_{ORS}}{t_{OPT}} \leq \frac{3l \cdot n^2 + 3l \cdot n + \lceil a \rceil}{\lceil a \rceil}$. Due to $\lceil a \rceil = (n-1) \cdot l + \lceil b \rceil$, the evacuate ratio of ORS is $\frac{3l \cdot n^2 + 4l \cdot n - (l - \lceil b \rceil)}{l \cdot n - (l - \lceil b \rceil)}$.

For any $n, l, \lceil a \rceil, \lceil b \rceil \in N*$, we can obtain $c = \frac{3n^2 \cdot l + 4n \cdot l - 3(l - \lceil b \rceil)}{n \cdot l - (l - \lceil b \rceil)}$. Assume $c = f(\lceil b \rceil)$, $\frac{df(\lceil b \rceil)}{d\lceil b \rceil} = \frac{-3n^2 \cdot l - 3n \cdot l}{[(n \cdot l - l) + \lceil b \rceil]^2} < 0$.Therefore, the minimum value of evacuate ratio c about $\lceil b \rceil$ is $f(\lceil b \rceil)_{\min} = 3\frac{\lceil a \rceil}{l} + 4$ when $\lceil b \rceil = l$. The maximum value is $f(\lceil b \rceil)_{\max} = 3n + 7 + 3 \cdot \frac{2l - n - 2}{n \cdot l - l + 1}$ when $\lceil b \rceil = 1$. Furthermore, we obtain that $f(\lceil b \rceil)_{\max} < 3\frac{\lceil a \rceil}{l} + 7 + 3 \cdot \frac{2l - n - 2}{n \cdot l - l + 1} \leq 3\frac{\lceil a \rceil}{l} + 7 + 3(2l - 3) = 3\frac{\lceil a \rceil}{l} + 6l - 2$. According to Lemma 1, there is $c \leq 3\frac{\lceil a \rceil}{l} + 6l - 2 < 3\frac{\sqrt{2}R+1}{l} + 6l - 2$.

The Theorem follows. □

Particularly, evacuate ratio is less than $3\sqrt{2}R + 7$ when $l = 1$.

5.2.2 Case 2: Two Groups

In reality, evacuees may be divided into two groups to escape. The two groups can share information with each other by real-time communication device, such as cell phone. In this situation, we design a Two-group Reposition Strategy (as shown in Fig.7), whose description and analysis are as follows. In the strategy, l denotes the length of step unit, it can be any positive integer chosen by evacuees.

Two-Group Reposition Strategy (TRS)

Step1: $t = 0$, $i = 1$, $j = 0$, evacuees are divided into two groups G_1 and G_2 at point O. Let $D = \{D_0, D_1\} = \{+\overrightarrow{x}(+\overrightarrow{y}), -\overrightarrow{x}(-\overrightarrow{y})\}$.

Step2: $G_1(G_2)$ walks toward D_j with length of $i \cdot l$. If they reach the boundary of M, skip to Step 3. Otherwise, retrace back to point O, $i = i+1$, $j = (-j)+1$.

Step3: If $G_1(G_2)$ has reached the boundary of M, $G_1(G_2)$ retrace back to point O and according to final direction of $G_1(G_2)$ to region boundary.

Theorem 5. *The evacuate ratio of TRS is less than* $\frac{\sqrt{2}R+1}{l} + l + 3$.

Proof. According to TRS, we divide it into two stages: The first stage, two groups of evacuees are both in convex region; and the second stage, at least one group is out of the convex region.

Similarly, for any $(n-1) \cdot l < \lceil a \rceil \leq n \cdot l$ where $n \in N*$, evacuation time of TRS is $t_{TRS1} \in [\lceil a \rceil + l \cdot n^2 - l \cdot n, l \cdot (n+1)^2 - l - (l \cdot n - \lceil a \rceil)]$ in first stage and $t_{TRS2} \in [0, 2\lceil a \rceil]$ in second stage. Thus there is $c = \frac{t_{TRS1}+t_{TRS2}}{t_{OPT}} \leq \frac{l\cdot(n+1)^2-l-(l\cdot n-\lceil a \rceil)+2\lceil a \rceil}{\lceil a \rceil}$. Due to $\lceil a \rceil = (n-1) \cdot l + \lceil b \rceil$, the evacuate ratio of TRS is $\frac{n^2\cdot l+4n\cdot l-3(l-\lceil b \rceil)}{n\cdot l-(l-\lceil b \rceil)}$. Similar with Theorem 3, for any $n, l, \lceil a \rceil, \lceil b \rceil \in N*$, the evacuate ratio of TRS is less than $\frac{\sqrt{2}R+1}{l} + l + 3$.

The Theorem follows. □

Particular, evacuate ratio is less than $\sqrt{2}R + 5$ when $l = 1$.

5.2.3 Case 3: Four Groups

In this part, we analyze the evacuation of four groups with complete information sharing with each other. When evacuees form four groups in grid network, we still use equally divided exploration strategy of the general network according to four directions of path in grid network (as shown in Fig.8). Based on assumptions of grid network, we reduce the evacuate ratio to 3.

Theorem 6. *In grid network, the evacuate ratio of EDES with four groups is 3.*

Proof. According to the strategy, four groups of evacuees escape along the four directions of grid network. Therefore, evacuation path of the first arrival at the boundary of convex region is the same as the shortest path. When $t = t_i$, suppose that evacuee G_1 reaches the boundary of convex region first. According to

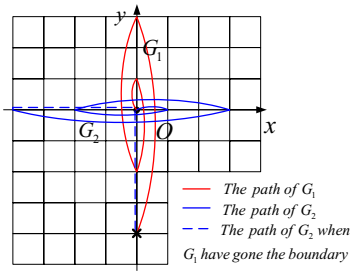


Fig. 7. The TRS when $l = 1$

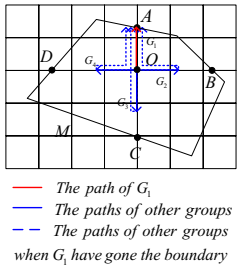


Fig. 8. The EDES of four groups

Theorem 3, evacuation time of G_1 is $\lceil a \rceil$. Therefore, evacuation time of all the evacuees is $3\lceil a \rceil$. Thus, the evacuate ratio of this strategy is $c = \frac{3\lceil a \rceil}{\lceil a \rceil} = 3$.

The Theorem follows. □

The evacuate ratio of EDES of four groups in grid network matches the lower bound according to Corollary 1. Thus, the EDES is optimal.

6 Comparison

In this section, we analyze the performance of strategies given in this paper by comparing them in different situations.

6.1 Comparison of Different Scenarios

To analyze the evacuation problem in this paper, we consider two scenarios: in general plane and in grid network respectively, as shown in Table 1.

Table 1. The evacuate ratio of different group numbers in different scenarios

	$k = 3$	$k = 4$	$k = 5$	$k = 6$
General plane	6	$3\sqrt{2} \approx 4.24$	$\frac{3}{\cos(\pi/5)} \approx 3.71$	$2\sqrt{3} \approx 3.46$
Grid network	-	3	3	3

Through horizontal and longitudinal comparison, we can conclude several properties from the table. First, the performance of strategy in general plane becomes more and more efficient with the number of groups increasing, which implies that communication among different groups is helpful. When the number of groups in grid network is more than four, the evacuate ratio of strategy is 3, which implies the evacuate ratio in four groups already matches the lower bound, and increasing the number of groups is not helpful. Secondly, the performance of strategy in grid network is better than that in general plane. In gird network, evacuees have more information about network feature. With the structure of network being known by evacuees, it makes them more easily to find the shortest path because the number of paths is constrained by structure of network.

6.2 Comparison of Different Cases

We compare the evacuate ratio among different cases in grid network, as shown in Table 2. From analysis in section 5 above, it is indicated that the $\frac{\lceil a \rceil}{l}$ makes sense to the performance of strategies, which denotes degrees of their prediction over step length. The evacuees in emergency tend to predict optimal solution $\lceil a \rceil$. When the evacuees evacuate from affected area, they will choose a specific l as step length in their prediction of the optimal solution.

Table 2. The evacuate ratio in different cases

$l = 1$	$\lceil a \rceil / l$ **is unknown**	$\lceil a \rceil / l = 1$	$\lceil a \rceil / l = 2$	$\lceil a \rceil / l = 3$
One group	$3\sqrt{2}R + 7$	7	10	13
Two groups	$\sqrt{2}R + 5$	5	6	7
Four groups	3	3	3	3

Through the comparison of strategies in different situations as shown above, it can be concluded that evacuees are wiser when $\frac{\lceil a \rceil}{l} = 1$. That is, if their prediction is equal to the step length they choose, the performance of strategies is better than other. It is also clear that the performance of strategies is improved by the number of groups increasing. Due to this, we can conclude that the performance of strategy is better with more information. This information may come from not only the communication with other groups of evacuees but also the prediction of the optimal solution.

7 Conclusion

In this paper, we studies the evacuation problem from a convex region in the plane during emergency situation in case that boundary information is unavailable. We prove the lower bound of this problem is 3 and analyses this problem in two scenarios. In general plane, we design EDES strategy of multiple groups and prove the evacuate ratio of the strategy is $\frac{3}{\cos(\pi/k)}$ which is closed to lower bound when the number of groups is numerous. In grid network, we analyze evacuation problems in different situations. Under the situations of one group and two groups, we design reposition strategies and analyze evacuate ratio of strategy respectively. Under four groups of evacuees, we prove that EDES in grid network is optimal. Moreover, we compare performance of different situations and indicate information is helpful.

Two extensions of this paper can be considered in future. One is the evacuation situation which different evacuees are located in different points in the convex region. The other consideration is the evacuation strategy with the information of affected area partially known.

Acknowledgments. The authors would like to acknowledge the financial support of Grants (No. 71071123 and 61221063.) from NSF of China and (No.IRT1173) from PCSIRT of China.

References

1. Deng, X.T., Kameda, T., Papadimitriou, C.: How to learn an unknown environment I: The rectilinear case. Journal of the ACM 45(2), 215–245 (1998)
2. Burgard, W., Moors, M., Fox, D., Simmons, R., Thrun, S.: Collaborative multi-robot exploration. In: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings, vol. 1, pp. 476–481 (2000)
3. Fleischer, R., Kamphans, T., Klein, R., Langetepe, E., Trippen, G.: Competitive online approximation of the optimal search ratio. Siam Journal on Computing 38(3), 881–898 (2008)
4. Batta, R., Chiu, S.S.: Optimal Obnoxious Paths on a Network: Transportation of Hazardous Materials. Operations Research 36, 84–92 (1988)
5. Panaite, P., Pelc, A.: Exploring unknown undirected graphs. Journal of Algorithms 33(2), 281–295 (1999)
6. Papadimitriou, C.H., Yannakakis, M.: Shortest path without a map. Theoretical Computer Science 84(1), 127–150 (1991)
7. Chen, X., Zhan, F.B.: Agent-based simulation of evacuation strategies under different road network structures. Journal of the Operational Research Society 59, 25–33 (2008)
8. Hamacher, H.W., Tjandra, S.A.: Mathematical modeling of evacuation problems: A state of the art. Pedestrian and Evacuation Dynamics, 227–266 (2002)
9. Lu, Q., George, B., Shekhar, S.: Capacity constrained routing algorithms for evacuation planning: A summary of results. In: Medeiros, C.B., Egenhofer, M., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 291–307. Springer, Heidelberg (2005)
10. Shekhar, S., Yang, K., Gunturi, V.M.V., Manikonda, L., et al.: Experiences with evacuation route planning algorithms. International Journal of Geographical Information Science 26(12), 2253–2265 (2012)
11. Berman, P.: On-line searching and navigation. In: Fiat, A. (ed.) Online Algorithms 1996. LNCS, vol. 1442, pp. 232–241. Springer, Heidelberg (1998)
12. Zhang, H., Xu, Y.: The k-canadian travelers problem with communication. In: Atallah, M., Li, X.-Y., Zhu, B. (eds.) FAW-AAIM 2011. LNCS, vol. 6681, pp. 17–28. Springer, Heidelberg (2011)

Kernelization and Lower Bounds of the Signed Domination Problem*

Ying Zheng, Jianxin Wang, and Qilong Feng

School of Information Science and Engineering
Central South University
Changsha 410083, P.R. China

Abstract. A function $f : V \to \{-1, +1\}$ defined on the vertex set of a graph $G = (V, E)$ is a *signed dominating function* if the sum of its function values over the closed neighborhood of every vertex is positive. A *signed dominating set* of a graph G (with respect to a signed dominating function f) is the set of vertices in G that are assigned value $+1$ by f. The minimum signed dominating set problem has important applications in the study of social networks. In this paper, we present a general technique that can be used to obtain kernels for the signed dominating set problem on various graphs. Our kernelization results also lead to strong and tight lower bounds on the size of minimum signed dominating sets and signed domination numbers for many graph classes.

1 Introduction

Let $G = (V, E)$ be a graph. For each vertex $v \in V$, let $N(v)$ be the set of all neighbors of v, and let $N[v] = N(v) \cup \{v\}$. A function $f : V \to \{0, 1\}$ is a *dominating function* if $\sum_{w \in N[v]} f(w) \geq 1$ for every vertex v in G. The famous dominating set problem can be defined in terms of dominating functions. A *dominating set* for the graph G (associated with a dominating function f) is the set of vertices in G that are assigned value 1 by the function f. The dominating set problem has been extensively studied in the literature (see, for example, [11]).

A variation of the dominating set problem is the *signed dominating set* problem. A function $f : V \to \{-1, +1\}$ is a *signed dominating function* for a graph $G = (V, E)$ if $\sum_{w \in N[v]} f(w) \geq 1$ for every vertex v in G. A *signed dominating set* for the graph G (associated with a signed dominating function f) is the set of vertices in G that are assigned value $+1$ by the function f. The *weight* $w(f)$ of a signed dominating function f for a graph G is defined to be $\sum_{v \in V} f(v)$. The *signed domination number* $\gamma_s(G)$ of the graph G is defined to be the minimum $w(f)$ over all dominating functions f for G.

There is a variety of applications for signed dominating sets. Assigning values -1 or $+1$ to the vertices of a graph can be modeled as networks of positive and negative electrical charges, networks of positive and negative spins of electrons,

* Supported by the National Natural Science Foundation of China under Grant (61232001, 61103033, 61173051).

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 261–271, 2013.

and networks of people or organizations in which global decisions must be made (e.g. yes-no, agree-disagree, like-dislike, etc.). In such a context, for example, the minimum number of vertices assigned value $+1$ by a signed dominating function represents the minimum number of people whose positive votes can dominate all local groups (represented by vertex neighborhoods in the graph), even though the entire network may have many negative voters. Hence this variation of domination studies situations in which, in spite of the presence of negative vertices, the neighborhoods of all vertices are required to maintain a positive sum.

The concept of signed domination in graphs was introduced by Dunbar *et al.* [6]. In particular, the complexity of computing the minimum weight of signed dominating functions for a graph has been studied by many researchers [6,23,9,14,22,16]. The decision version of the problem of computing the signed domination number γ_s of a graph is NP-complete, even when the graph is restricted to chordal graph or bipartite graph [10]. For a fixed k, the problem of determining if a graph has a signed dominating function of weight at most k is also NP-complete [10]. A linear-time algorithm for finding a signed dominating function of the minimum weight in an arbitrary tree was presented by Hattingh *et al.* [10]. Much research on signed domination has been focused on deriving better upper and lower bounds on the signed domination number γ_s of graphs. Dunbar *et al.* [6] investigated the properties of the signed domination number and established lower bounds for γ_s: for r-regular n-vertex graphs, when r is even, $\gamma_s \geq n/(r+1)$. Henning and Slater [13] pointed out that $\gamma_s \geq 2n/(r+1)$ when r is odd. Upper bounds were given by Henning [12] and Favaron [7]: when r is odd, $\gamma_s \leq n(r+1)^2/(r^2+4r-1)$, and when r is even, $\gamma_s \leq n(r+1)/(r+3)$. Haynes[11] put forward that for any graph if $\Delta \leq 3$, $\gamma_s \geq n/3$, and $\Delta \leq 5$, $\gamma_s \geq 0$, where Δ is the maximum vertex degree of the graph. Zhang *et al.*[22] proposed that for any graph $G = (V, E)$, $n = |V|$ and $m = |E|$, then $\gamma_s \geq n - 2m/3$. More recently, Fernau and Rodriguez-Velazquez [8] studied the relationship between graph signed dominations and other related graph parameters.

Traditional method of computing the signed domination number γ_s is so complicated since it needs a lot of energy to analyze the vertex degree and the edge relationship between the vertices assigned value $+1$ and -1 by signed dominating function(see [7,23,21]). Specifically, only the lower bounds of γ_s on general graphs, r-regular graphs and $\Delta \leq 5$ graphs are known before our work. Moreover, the corresponding lower bounds are usually not sharp. Chen et al.[21] give a complicated proof to get the lower bound of the signed domination number on graphs with minimum degree $\delta \geq 2$ and maximum degree Δ. For grids, one of the most challenging problems concerning the domination numbers of Cartesian products of graphs is the proof of the Vizing Conjecture[18], namely $\gamma(G \times H) \geq \gamma(G) \cdot \gamma(H)$. Partial works have been made towards finding the domination numbers of some particular Cartesian product of graphs. For example, the domination number has been bounded both above and below for square grids, [15]. In short, no result has been known for the lower bounds of γ_s on planar graphs, d-partite graphs, bipartite graphs and Cartesian products of graphs so far.

In this paper, we develop a framework for systematically deriving kernels for signed dominating set problem on general and other special graphs. Our technique applies to a series of graphs. Kernels of those graphs yield the signed domination number γ_s quickly and easily. Furthermore, the lower bounds of γ_s we derived are tight.

> (Parameterized) Signed Dominating Set. Given a graph G and a positive integer k, determine if there exists a signed dominating set of at most k vertices in G.

We will study the complexity of the Signed Dominating Set problem and present a number of kernelization results for the problem. We first study the Signed Dominating Set problem and show that the problem is NP-complete even restricted to bipartite graphs, bounded-degree graphs, planar graphs, grids and many other graph classes. Then we give kernelization results for the Signed Dominating Set problem on the following graph classes: general graphs, planar graphs, grid graphs, d-partite graphs, bipartite graphs, bounded-degree graphs and r-regular graphs. Our kernelization results also lead to strong tight lower bounds, on the size of minimum signed dominating sets for these graph classes. These results also imply strong lower bounds on the signed domination number γ_s of these graph classes.

Our results are summarized in Fig. 1.

Graph Class	Kernel	Signed Domination Number
general graphs	$(k^2+k)/2$	$\sqrt{8n+1}-n-1$
d-partite graphs	$\frac{d-1}{2d}k^2+k$	$\frac{2d}{d-1}(\sqrt{1+\frac{2(d-1)}{d}n}-1)-n$
bipartite graphs	$\lfloor k/2\rfloor\cdot\lceil k/2\rceil+k$	$4(\sqrt{n+1}-1)-n$
bounded-degree graphs	$(\lfloor\Delta/2\rfloor+2)k/2$	$\frac{2-\lfloor\Delta/2\rfloor}{2+\lfloor\Delta/2\rfloor}n$
planar graphs	$4k-6$	$(6-n)/2$
r-regular graphs	$(r+1)k/(\lceil r/2\rceil+1)$	$\frac{2\lceil r/2\rceil-r+1}{r+1}n$
grid graphs	$5k/3+4$	$(n-24)/5$

Fig. 1. Summary of our results for Signed Dominating Set (where Δ denotes the maximum vertex degree of the graph)

2 Preliminaries

For a set S, we will denote by $|S|$ the cardinality of S. Let $G=(V,E)$ be a graph. We will denote by $V(G)$ and $E(G)$ the vertex set and the edge set of the graph G, respectively. For a vertex v in G, the sets $N(v)$ and $N[v]$ will be called the *open neighborhood* and the *closed neighborhood* of v, respectively. The *degree* of the vertex v, $deg_G(v)$, is equal to $|N(v)|$. A graph is a *planar graph* if it can be embedded in the plane without edge crossing. A graph $G=(V,E)$ is called

d-partite if V admits a partition into d classes such that every edge has its ends in different classes: vertices in the same partition class must not be adjacent. A graph is a *bipartite graph* if $d = 2$. A *triangle-free* graph is a graph containing no cycles of length three. A graph in which every vertex has the same degree is called a *regular graph*. If every vertex has degree r then we say the graph is *r-regular graph*. A graph class is of *bounded-degree* if there is a constant c such that every graph in the class have their vertex degree bounded by c. For any two graphs G and H, the Cartesian product $G \times H$ is the graph with vertex set $V(G) \times V(H)$ and with edge set $E(G \times H)$ such that $(u_1, v_1), (u_2, v_2) \in E(G \times H)$, whenever $v_1 = v_2$ and $u_1u_2 \in E(G)$, or $u_1 = u_2$ and $v_1v_2 \in E(H)$. The numbers $0, 1, 2, ..., n-1$ always denote the vertices of the path P_n or the cycle C_n. The graph $P_m \times P_n$ is called *grid* graph, and the graph $P_m \times C_n$ can be termed as *cylindrical grid* graph as shown in Fig. 2.

3 The Complexity of Signed Dominating Set

It has been known that the problem of determining if a given graph G has a signed dominating function of weight k is NP-complete [10]. It is easy to see that given the size of a minimum signed dominating set of a graph, we can easily compute the signed domination number for the graph. Therefore, we derive from [10] immediately that the minimum signed dominating set problem is NP-hard. On the other hand, the size of a minimum signed dominating set and the value of the signed domination number can differ very significantly. We first study the relationship between the weight of dominating functions and the size of dominating sets for a graph.

Lemma 1. *Let G be a graph of n vertices such that G has a signed dominating function f of weight k. Then the integers n and k have the same parity.*

Proof. Suppose that the signed dominating function f assigns h of the n vertices in G with value $+1$ and assigns the remaining $n-h$ vertices in G with value -1. Then we have the equation:

$$h - (n - h) = k,$$

which shows that $n + k = 2h$ is an even number. As a consequence, the integers n and k must have the same parity. □

Using Lemma 1, we can immediately derive the following result.

Lemma 2. *A graph G of n vertices has a signed dominating function f of weight k if and only if it has a signed dominating set of $\lceil (n-1)/2 \rceil + \lfloor (k+1)/2 \rfloor$ vertices.*

Proof. By Lemma 1, we can divide the proof into two cases: either both n and k are even numbers or both n and k are odd numbers.

Suppose that both n and k are even numbers. Let f_1 be a signed dominating function of weight k for the graph G. Let h be the number of vertices in G that

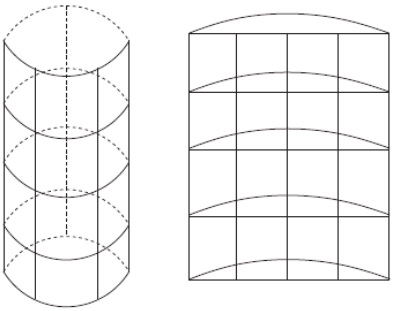


Fig. 2. Two different looks of the cylindrical grid graph $P_5 \times C_5$

are assigned value $+1$ by f_1. Then the number of vertices in G that are assigned value -1 by f_1 is equal to $n-h$. Since the weight of f_1 is k, we get $h-(n-h) = k$, which gives (note that both n and k are even):

$$h = (n+k)/2 = n/2 + k/2 = \lceil (n-1)/2 \rceil + \lfloor (k+1)/2 \rfloor.$$

Therefore, the graph G has a signed dominating set of $\lceil (n-1)/2 \rceil + \lfloor (k+1)/2 \rfloor$ vertices. On the other hand, if the graph G has a signed dominating set of $\lceil (n-1)/2 \rceil + \lfloor (k+1)/2 \rfloor = n/2 + k/2$ vertices, then G has a signed dominating function f_2 that assigns value $+1$ to $n/2 + k/2$ vertices and assigns value -1 to $n - (n/2 + k/2) = n/2 - k/2$ vertices. The weight of this signed dominating function f_2 is then equal to $(n/2 + k/2) - (n/2 - k/2) = k$.

Now consider the case when both n and k are odd numbers. Let f_3 be a signed dominating function of weight k for the graph G. Suppose that f_3 assigns value $+1$ to h vertices and value -1 to $n - h$ vertices. We get $h - (n - h) = k$, which gives (note that both n and k are odd):

$$h = (n+k)/2 = (n-1)/2 + (k+1)/2 = \lceil (n-1)/2 \rceil + \lfloor (k+1)/2 \rfloor,$$

i.e., the graph G has a signed dominating set of $\lceil (n-1)/2 \rceil + \lfloor (k+1)/2 \rfloor$ vertices. On the other hand, if the graph G has a signed dominating set of $\lceil (n-1)/2 \rceil + \lfloor (k+1)/2 \rfloor = (n-1)/2 + (k+1)/2 = (n+k)/2$ vertices, then G has a signed dominating function f_4 that assigns value $+1$ to $(n+k)/2$ vertices and assigns value -1 to $n - (n+k)/2 = (n-k)/2$ vertices. The weight of this signed dominating function f_4 is then equal to $(n+k)/2 - (n-k)/2 = k$.

This completes the proof of the lemma. □

Combining Lemma 2 with the results in [10], we get immediately that the problem of determining if a given graph has a signed dominating set of size k is NP-complete.

Next we show that Signed Dominating Set problem is NP-complete for grid graphs.

Lemma 3. *[17] A planar graph G with maximum degree 4 can be embedded in the plane using $O(|V|)$ area in such a way that its vertices are at integer coordinates and its edges are drawn so that they are made up of line segments of the form $x = i$ or $y = j$, for integers i and j.*

Theorem 1. *The* Signed Dominating Set *problem is NP-complete even for any subgraphs of the grid graph $P_j \times P_l$, $j \geq l \geq 3$.*

Proof. Deciding a signed dominating function of weight less than k on planar graph of maximum degree 3 is known to be NP-complete[4]. Using Lemma 2, we conclude that Planar Signed Dominating Set Problem of maximum degree 3 is NP-complete. The reduction is from Planar Signed Dominating Set of maximum degree 3. We transform a planar graph $G = (V, E)$ of maximum degree 3 into grid graph G' such that G has a signed dominating set D with $|D| \leq k$ if and only if G' has a signed dominating set D' with $|D'| \leq k'$.

By Lemma 3 we can embed G in the plane with line segments parallel to the $x-$ or $y-$ axis. It is an easy matter ensure that each line segment has integer length, and that the total line length for the line representing an edge (u, v) is of the form $2k_{uv} + 1$ for some integer k_{uv}. A grid graph G' is induced by this drawing, containing exactly those integer points lying on a line in the drawing. The construction of G' can clearly be accomplished in polynomial time. It is easy to verify that there exists a signed dominating set D in G with $|D| \leq k$ if and only if there exists a signed dominating set D' in G' with $|D'| \leq k + \sum_{uv \in E(G)} k_{uv}$. □

4 Kernelizations of Signed Dominating Set

In this section, we consider kernelization algorithms for the Signed Dominating Set problem on various graph classes. We introduce a systematic method that allows us to derive strong lower bounds on the size of a minimum signed dominating set of a graph for many graph classes, and to develop kernelization algorithms that produce small kernels for the Signed Dominating Set problem on these graph classes.

In the entire discussion of this section, we will always assume that our graphs are simple graphs. Thus, multiple edges and self-loops are not allowed in a graph.

We start with the following simple observations.

Lemma 4. *Let (G, k) be an instance of* Signed Dominating Set*. If the graph G has a vertex of degree at least $2k$, then (G, k) is a no-instance.*

Proof. If a vertex v has degree at least $2k$, then for a set D of vertices in G to satisfy the signed dominating condition $\sum_{w \in N[v]} f_D(w) \geq 1$, the subset $N[v]$ must have at least $k + 1$ vertices w with $f_D(w) = +1$. Thus, there cannot be a subset of at most k vertices in G that makes a signed dominating set. □

Lemma 5. *Suppose that a graph G has a signed dominating set D. Then every vertex v that is not in D has at least two neighbors in D.*

Proof. Let f_D be the signed dominating function corresponding to the signed dominating set D. Let v be a vertex of G that is not in D. If v has less than two neighbors in the set D, then the signed dominating relation $\sum_{w \in N[v]} f_D(w) \geq 1$ cannot be satisfied. Thus, the vertex v must have at least two neighbors in the set D. □

The following lemma is critical for our development of the kernelization algorithms for the Signed Dominating Set problem. The lemma is based on a technique that is a modification of the technique recently introduced in [19].

Lemma 6. *Let D be a signed dominating set of a simple graph G, and let $G[D]$ be the subgraph induced by the set D. Then the number of vertices in the graph G is bounded by $|E(G[D])| + |D|$, where $E(G[D])$ is the set of edges in the induced subgraph $G[D]$.*

Proof. Let $G = (V, E)$ and let f_D be the signed dominating function corresponding to the signed dominating set D. Let J be the set of vertices w in G with $f_D(w) = -1$, i.e., $J = V \setminus D$. For a vertex $v \in D$, we will denote by $deg_{G[D]}(v)$ the degree of the vertex v in the induced subgraph $G[D]$. By Lemma 5, each vertex w in J has at least two neighbors in the set D.

If a vertex v in D has h neighbors in J, then the vertex v must have at least h neighbors in D in order to satisfy the signed dominating relation $\sum_{u \in N[v]} f_D(u) \geq 1$. That is, $deg_{G[D]}(v) \geq h$.

Let D_h be the subset of D such that each vertex in D_h has exactly h neighbors in J (note that if $D_h \neq \emptyset$ then $h \leq |D|-1$). Now we count the number of vertices in J in terms of the neighborhood of vertices in D, and we will get the following relations (note that each vertex in J has at least two neighbors in D):

$$|J| \leq \frac{1}{2} \sum_{h=0}^{|D|-1} h \cdot |D_h| \leq \frac{1}{2} \sum_{h=0}^{|D|-1} \sum_{v \in D_h} deg_{G[D]}(v) = \frac{1}{2} \sum_{v \in D} deg_{G[D]}(v) = |E(G[D])|.$$

Therefore, the total number $|V|$ of vertices in the graph G satisfies

$$|V| = |J| + |D| \leq |E(G[D])| + |D|.$$

This completes the proof of the lemma. □

Lemma 6 allows us to derive strong lower bounds on the size of a minimum signed dominating set and develop kernelization algorithms for the Signed Dominating Set problem on a number of graph classes.

4.1 Kernel for General Graphs

We start with the Signed Dominating Set problem on general graphs.

Theorem 2. *The* Signed Dominating Set *problem on general graphs admits a kernel of at most $(k^2 + k)/2$ vertices.*

Proof. The kernelization algorithm on an instance (G, k) of Signed Dominating Set on general graphs runs as follows:

1. **if** (G has a vertex of degree at least $2k$) or (G contains more than $(k^2+k)/2$ vertices)
2. **then** return a no-instance (G', k'), where G' is a star with 4 leaves and $k' = 3$;
3. **else** return $(G', k') = (G, k)$.

Because of the conditions enforced in step 1, in the instance $(G', k') = (G, k)$ returned by step 3 of the algorithm, the graph G' contains at most $((k')^2 + k')/2$ vertices.

Now consider step 2 of the algorithm. By Lemmas 4, if the graph G has a vertex of degree at least $2k$, then (G, k) must be a no-instance. If G has a signed dominating set D with $h \leq k$ vertices, then the induced subgraph $G[D]$ has at most $h(h-1)/2$ edges (note that G is a simple graph), i.e., $|E(G[D])| \leq h(h-1)/2$. By Lemma 6, the graph G has at most

$$|E(G[D])| + |D| \leq h(h-1)/2 + h = h(h+1)/2 \leq (k^2+k)/2$$

vertices. Therefore, if G has more than $(k^2+k)/2$ vertices, then (G, k) must be a no-instance. Note that the instance (G', k') returned in step 2 is a no-instance in which the graph G' contains 5 vertices and $5 \leq ((k')^2 + k')/2 = 6$.

Thus, in the instance (G', k') returned by the algorithm, the graph G' has at most $((k')^2+k')/2$ vertices, and is a yes-instance of Signed Dominating Set if and only if the original instance (G, k) is a yes-instance of Signed Dominating Set. □

Corollary 1. *Let G be a simple graph of n vertices. Then a signed dominating set for G contains at least $(\sqrt{8n+1}-1)/2$ vertices.*

Proof. Let D be a signed dominating set for G with $|D| = k$. By Lemma 6, $n \leq (k^2+k)/2$. Solving this inequality, we get $k \geq (\sqrt{8n+1}-1)/2$. □

The lower bound given in Corollary 1 is tight. This can be seen from the following example. Start from a complete graph K_h of h vertices, $h \geq 4$. For each pair (u, v) of vertices in K_h, add a new degree-2 vertex w_{uv} whose two neighbors are u and v. Let the resulting graph be G. The graph G has $n = h(h-1)/2+h = h(h+1)/2$ vertices. It is easy to verify that the h vertices in the original K_h make a signed dominating set for the graph G, and $h = (\sqrt{8n+1}-1)/2$.

4.2 Kernel for Planar Graphs

Now we consider the Signed Dominating Set problem on planar graphs.

Lemma 7. *Let G be a planar graph that has a signed dominating set D with k vertices. Then G has at most $4k-6$ vertices.*

Proof. By Lemma 6, the graph G has at most $|E(G[D])| + |D|$ vertices. Since G is a planar graph, the induced subgraph $G[D]$ is also a planar graph. By a well-known relationship between the number of vertices and the number of edges in a planar graph [20], we have $|E(G[D])| \leq 3|V(G[D])| - 6 = 3|D| - 6$. Replacing $|D|$ with k, we get immediately that the graph G has at most $4k-6$ vertices. □

The following theorem now follows directly from Lemma 7.

Theorem 3. *The* Signed Dominating Set *problem on planar graphs admits a kernel of at most* $4k - 6$ *vertices.*

Corollary 2. *Let* G *be a planar graph of* n *vertices. Then a signed dominating set of* G *contains at least* $(n + 6)/4$ *vertices.*

The lower bound given in Corollary 2 is tight. To see this, we start with a triangulated planar embedding of a graph H of h vertices (H has exactly $3h-6$ edges). Then for each edge (u, v) in H add a new degree-2 vertex w_{uv} whose two neighbors are u and v. The resulting graph G is a planar graph with $n = 4h-6$ vertices, and the $h = (n + 6)/4$ vertices in the original graph H make a signed dominating set for the graph G.

4.3 Kernel for d-Partite Graphs

Lemma 8. *Let* G *be a* d*-partite graph that has a signed dominating set* D *with* k *vertices. Then* G *has at most* $\frac{d-1}{2d}k^2 + k$ *vertices.*

4.4 Kernel for Bounded-Degree Graphs

Lemma 9. *Let* G *be a graph in which the maximum vertex degree is* Δ*. Suppose that the graph* G *has a signed dominating set* D *with* k *vertices. Then the graph* G *has at most* $(\lfloor \Delta/2 \rfloor + 2)k/2$ *vertices.*

4.5 Kernel for r-Regular Graphs

Lemma 10. *Let* G *be an* r*-regular graph in which each vertex with degree exactly* r*. Suppose that the graph* G *has a signed dominating set* D *with* k *vertices. Then the graph* G *has at most*

$$\begin{cases} \frac{2r+2}{r+2}k & r \text{ is even} \\ \frac{2r+2}{r+3}k & r \text{ is odd} \end{cases}$$

vertices. That is G *has at most* $\frac{r+1}{\lceil r/2 \rceil + 1}k$ *vertices.*

4.6 Kernel for Grid Graphs

Lemma 11. *Let* $P_j \times P_l$ *be a grid graph with* $j \geq 3$ *and* $l \geq 3$*. Suppose that the grid* $P_j \times P_l$ *has a signed dominating set* D *with* k *vertices. Then the graph* G *has at most* $5k/3 + 4$ *vertices.*

5 Conclusion

In this paper, we study the Signed Dominating Set problem from the parameterized perspective. We proved the NP-completeness for the problem on various graph classes, developed kernelization results for the problem on various graph classes. Our kernelization results also lead to strong lower bounds, most of them are tight, on the size of minimum signed dominating set for these graph classes. Since the size $\theta_s(G)$ of a minimum signed dominating set for a graph G of n vertices and the signed domination number $\gamma_s(G)$ of the graph G are related by the equality $\gamma_s(G) = 2\theta_s(G) - n$, our strong lower bounds on θ_s also lead to strong lower bounds on γ_s. We list these results as follows:

- The signed domination number $\gamma_s(G)$ of a general graph G of n vertices is at least $\sqrt{8n+1} - n - 1$, and this bound is tight.
- The signed domination number $\gamma_s(G)$ of a planar graph G of n vertices is at least $(6-n)/2$, and this bound is tight.
- The signed domination number $\gamma_s(G)$ of a graph G of n vertices in which the maximum vertex degree is Δ is at least $(4-\Delta)n/(\Delta+4)(\Delta$ is even) or $(5-\Delta)n/(\Delta+3)(\Delta$ is odd) and this bound is tight.
- The signed domination number $\gamma_s(G)$ of a bipartite graph G of n vertices is at least $4(\sqrt{n+5/4}-1)-n$, $4(\sqrt{n+1}-1)-n$ or $4(\sqrt{n+2}-1)-n$ and this bound is tight upper to a constant.
- The signed domination number $\gamma_s(G)$ of an r-regular graph G of n vertices is at least $n/(r+1)(r$ is even) or $2n/(r+1)(r$ is odd), and this bound is tight.
- The signed domination number $\gamma_s(G)$ of a d-partite graph G of n vertices is at least $\frac{2d}{d-1}(\sqrt{1+\frac{2(d-1)}{d}n}-1)-n$.
- The signed domination number $\gamma_s(G)$ of a grid graph G of n vertices is at least $(n-24)/5$.

The Signed Dominating Set problem seems quite different from the regular Dominating Set problem. For example, the Dominating Set problem on general graphs is $W[2]$-hard [5] while the Signed Dominating Set problem on general graphs has a quadratic kernel (Theorem 2), which implies immediately that the problem is fixed-parameter tractable. Moreover, the kernel of $4k-6$ vertices for Signed Dominating Set on planar graphs (Theorem 3) seems relatively easier, compared to the highly nontrivial algorithms for developing linear kernels for Dominating Set on planar graphs [1,3].

References

1. Alber, J., Fellows, M., Niedermeier, R.: Polynomial time data reduction for dominating set. J. ACM 51, 363–384 (2004)
2. Bollobás, B.: Modern Graph Theory. Springer, New York (1998)
3. Chen, J., Fernau, H., Kanj, I., Xia, G.: Parametric duality and kernelization: lower bounds and upper bounds on kernel size. SIAM J. Comput. 37, 1077–1106 (2007)

4. Damaschke, P.: Minus domination in small-degree graphs. Discrete Applied Mathematics 108, 53–64 (2001)
5. Downey, R., Felows, M.: Parameterized Complexity. Springer, New York (1999)
6. Dunbar, J., Hedetniemi, S., Henning, M., Slater, P.: Signed domination in graphs. Graph Theory, Combinatorics and Applications, 311–322 (1995)
7. Favaron, O.: Signed domination in regular graphs. Discrete Math. 158, 287–293 (1996)
8. Fernau, H., Rodriguez-Velazquez, J.: Alliances and related parameters in graphs (2012) (manuscript)
9. Hass, R., Wexler, T.: Signed domination numbers of a graph and its complement. Discrete Math. 283, 87–92 (2004)
10. Hattingh, J., Henning, M., Slater, P.: The algorithmic complexity of signed domination in graphs. Australasian Journal of Combinatorics 12, 101–112 (1995)
11. Haynes, T., Hedetniemi, S., Slater, P.: Domination in Graphs. Advanced Topics, vol. 2. CRC Press, New York (1998)
12. Henning, M.: Domination in regular graphs. Ars Combinatoria 43, 263–271 (1996)
13. Henning, M., Slater, P.: Irregularities relating domination parameters in cubic graphs. Discrete Math. 158, 87–98 (1996)
14. Matousek, J.: On the signed domination in graphs. Combinatorica 20, 103–108 (2000)
15. Nandi, M., Parui, S., Adhikari, A.: The domination numbers of cylindrical grid graphs. Applied Mathematics and Computation 217, 4879–4889 (2011)
16. Shan, E., Cheng, T.C.: Remarks on minus (signed) total domination in graphs. Discrete Mathematics 308, 3373–3380 (2008)
17. Valiant, L.G.: Universality considerations in VLSI circuits. IEEE Trans. Computers 30, 135–140 (1991)
18. Vizing, V.G.: The Cartesian product of graphs. Vychisl Sistemy 9, 30–43 (1963)
19. Wang, J., Yang, Y., Guo, J., Chen, J.: Planar graph vertex partition for linear problem kernels. Journal of Computer and System Sciences (2012) (to appear)
20. West, D.: Introduction to Graph Theory, 2nd edn. Prentice-Hall (2000)
21. Zheng, Y., Wang, J., Feng, Q., Chen, J.: On parameterized complexity of the signed domination problem. Journal of Mathematical Structures in Computer Science (2012) (to appear)
22. Zhang, Z., Xu, B., Li, Y., Liu, L.: A note on the lower bounds of signed domination number of a graph. Discrete Math. 195, 295–298 (1999)
23. Zelinka, B.: Signed total domination number of a graph. Czechoslovak Math. 51, 225–229 (2001)

On Edge-Independent Sets

Ton Kloks*, Ching-Hao Liu*, and Sheung-Hung Poon*

Department of Computer Science
National Tsing Hua University, Taiwan
chinghao.liu@gmail.com, spoon@cs.nthu.edu.tw

Abstract. A set of edges in a graph G is *independent* if no two elements are contained in a clique of G. The *edge-independent set problem* asks for the maximal cardinality of independent sets of edges. We show that the edge-clique graphs of cocktail parties have unbounded rankwidth. There is an elegant formula that solves the edge-independent set problem for graphs of rankwidth one, which are exactly distance-hereditary graphs, and related classes of graphs. We present a PTAS for the edge-independent set problem on planar graphs and show that the problem is polynomial for planar graphs without triangle separators. The set of edges of a bipartite graph is edge-independent. We show that the edge-independent set problem remains NP-complete for graphs in which every neighborhood is bipartite, i.e., the graphs without odd wheels.

1 Introduction

A set of edges in a graph G is *independent* if no two elements are contained in a clique of G. The *edge-independent set problem* asks for the maximal cardinality of independent sets of edges, which is called the *edge-independence number* and denoted by $\alpha'(G)$.

Definition 1. *The edge-clique graph $K_e(G)$ of a graph G has the edges of G as its vertices and two vertices of $K_e(G)$ are adjacent when the corresponding edges in G are contained in a clique.*

The edge-independent sets of a graph G are the independent sets of the edge-clique graph of G, so we write $\alpha'(G) = \alpha(K_e(G))$, where $\alpha(H)$ denotes the cardinality of the maximum independent set of a graph H. There is a characterization of edge-clique graphs presented in [8], but the problem whether there is any polynomial-time recognition algorithm for edge-clique graphs remains open. Some results of edge-clique graphs can be found in [1, 7–9, 23, 32–34].

The edge-independence number is a lowerbound for the *edge-clique cover*. An edge-clique cover of G is a family of complete subgraphs such that each edge of G is in at least one member of the family. The *edge-clique cover problem* asks for the minimal number of such a family, which is called the *edge-clique cover number* and denoted by $\theta_e(G)$.

* Supported in part by grant nos. 101-2218-E-007-001 and 100-2628-E-007-020-MY3 of the National Science Council (NSC), Taiwan, R.O.C.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 272–283, 2013.

The Exponential Time Hypothesis asserts that any fixed-parameter algorithm for edge-clique cover problem must have a double exponential dependence on the parameter [12]. It is no surprise that the problem is difficult. Mathematicians have been trying to find orthogonal Latin squares for ages. The complete multipartite graph K_n^m with m partite sets of size n has an edge-clique cover with n^2 cliques if and only if there exists a collection of at least $m-2$ orthogonal Latin squares of order n [31]. The existence of 3 orthogonal Latin squares of order 10 is still wide open.

If the edge-clique graph of a graph G is perfect then the edge-independence number and edge-clique cover number coincide and they can be computed in polynomial time. Unfortunately, this happens rarely. For the cocktail party graphs, i.e., K_2^m, an independent set of edges contains only four elements, while the edge-clique cover is asymptotically $\log_2 m$. A formula is known for the edge-clique cover of cocktail parties [18]. For K_n^m with $n > 2$ the situation is highly unclear [12, 29].

We show that for cographs, which generalize cocktail parties, there exists a beautiful, elegant formula which enables us to compute the edge-independence number in linear time. In fact, the same formula holds for graphs of rankwidth one, i.e., distance-hereditary graphs.

Bipartite graphs are perfect, and the edge set of a bipartite graph is independent. An edge-clique cover for a bipartite graph is simply the set of edges, since no two edges are in a clique. We show that the edge-independence number is already NP-complete for graphs in which every neighborhood is bipartite, i.e., the graphs without odd wheels.

For general graphs, one of the major open problems is whether the edge-independence number can be computed in single exponential time, by which we mean $O^*(2^{O(n)})$ time, where n is the number of vertices in the graph. Our investigations indicate that the edge-independent set problem is 'much easier' than the edge-clique cover problem, however, we think that it is unlikely that there is a single exponential algorithm that solves it.

We organized this paper as follows. In Section 2 we show that edge-clique graphs of cocktail parties have unbounded rankwidth. Thus we emphasize the huge difference in complexity between the two problems for one of the easiest classes of graphs. In Section 3 we present our formula for the edge-independence numbers of cographs, distance-hereditary graphs and P_4-sparse graphs. Edge-clique cover is NP-complete for planar graphs.[1] In Section 4 we show that there is a polynomial-time approximation scheme (PTAS) for edge-independent set problem on planar graphs. In Subsection 4.1 we show that there is an efficient algorithm that computes the edge-independence number for planar graphs without triangle separator. In Section 5 we show that the edge-independent set problem remains NP-complete when every neighborhood is bipartite. We end the paper with some concluding remarks.

[1] Surprisingly, the problem to decide whether the edges of a planar graphs can be partitioned into triangles is polynomial [16].

2 Rankwidth of Edge-Clique Graphs of Cocktail Parties

In this section we show that the computation of edge-clique cover number $\theta_e(G)$ is probably much harder than the edge-independence number $\alpha'(G)$. For graphs of bounded rankwidth the edge-clique cover is polynomial-time computable [15].

Gyárfás [19] showed an interesting lowerbound in the following theorem. Here, two vertices x and y are *equivalent* if they are adjacent and have the same closed neighborhood.

Theorem 1. *If a graph G has n vertices and contains neither isolated nor equivalent vertices then $\theta_e(G) \geq \log_2(n+1)$.*

Notice that Gyárfás theorem implies that the edge-clique cover problem is fixed-parameter tractable.

Definition 2. *The cocktail party graph $cp(n)$ is the complement of a matching with $2n$ vertices.*

A cocktail party graph has no equivalent vertices. Thus, by Theorem 1,

$$\theta_e(cp(n)) \geq \log_2(2n+1).$$

For the cocktail party graph an exact formula for $\theta_e(cp(n))$ appears in [18]. In that paper Gregory and Pullman (see also [17, 24]) prove that

$$\lim_{n\to\infty} \frac{\theta_e(cp(n))}{\log_2(n)} = 1.$$

For a graph G we denote the vertex-clique cover number of G by $\kappa(G)$. Notice that, for a graph G,

$$\theta_e(G) = \kappa(K_e(G)).$$

Albertson and Collins mention the following result (due to Shearer) [1] for the graphs $K_e^r(cp(n))$, defined recursively by $K_e^r(cp(n)) = K_e(K_e^{r-1}(cp(n)))$.

$$\alpha(K_e^r(cp(n))) \leq 3 \cdot (2^r)!$$

Thus, for $r = 1$, $\alpha(K_e(cp(n))) \leq 6$. However, the following is easily checked (for a proof see Lemma 3).

Lemma 1. *For $n \geq 2$*

$$\alpha(K_e(cp(n))) = 4.$$

Definition 3. *A class of graphs $\mathcal{G}$ is χ-bounded if there exists a function f such that for every graph $G \in \mathcal{G}$,*

$$\chi(G) \leq f(\omega(G)),$$

where $\chi(G)$ is the chromatic number of G and $\omega(G)$ is the clique number of G.

Dvořák and Král prove that the class of graphs with rankwidth at most k is χ-bounded [13].

The following theorem illustrates that edge-clique graphs of cocktail parties have a 'hard structure.'

Theorem 2. *The class of edge-clique graphs of cocktail parties has unbounded rankwidth.*

Proof. It is easy to see that the rankwidth of any graph is at most one more than the rankwidth of its complement [30]. Assume that there is a constant k such that the rankwidth of $K_e(G)$ is at most k whenever G is a cocktail party graph. Let

$$\mathcal{K} = \{\, \overline{K_e(G)} \mid G \simeq cp(n),\ n \in \mathbb{N} \,\}.$$

Then the rankwidth of graphs in $\mathcal{K}$ is uniformly bounded by $k+1$. By the result of Dvořák and Král, there exists a function f such that

$$\log_2(2n+1) \leq \theta_e(G) = \kappa(K_e(G)) \leq f(\alpha(K_e(G))) = f(4)$$

for every cocktail party graph G. This is a contradiction. □

3 Algorithms for Distance-Hereditary Graphs and Related Graph Classes

First, we show that the edge-independent set problem is NP-complete for general graphs.

Lemma 2. *The computation of $\alpha'(G)$ for arbitrary graphs G is NP-complete.*

Proof. Let G be an arbitrary graph. Construct a graph H as follows. At every edge in G add two simplicial vertices, both adjacent to the two end vertices of the edge. Add one extra vertex x adjacent to all vertices of G. Let H be the graph constructed in this way.

Notice that a maximum set of independent edges does not contain any edge of G since it would be better to replace such an edge by two edges incident with the two simplicial vertices at this edge. Also notice that a set of independent edges incident with x corresponds with an independent set of vertices in G. Hence

$$\alpha'(H) = 2m + \alpha(G),$$

where m is the number of edges of G. □

Remark 1. To show that edge-independent set problem is $W[1]$-complete we would need a parameter-preserving reduction.

3.1 Cographs

A cograph is a graph without induced P_4, which is a path with four vertices. It is well-known that a graph is a cograph if and only if every induced subgraph with at least two vertices is either a join or a union of two smaller cographs. It follows that a cograph G has a decomposition tree (T, f) where T is a rooted binary tree and f is a bijection from the vertices of G to the leaves of T. Each internal node of T, including the root, is labeled as $\otimes$ or $\oplus$. The $\otimes$-node joins the two subgraphs mapped to the left and right subtree. The $\oplus$ unions the two subgraphs. When G is a cograph then a decomposition tree as described above can be obtained in linear time [11].

The neighborhood of a vertex x, which is denoted by $N(x)$, is the set of vertices that are adjacent to x. For a vertex x, let $d'(x)$ be the independence number of the subgraph of G induced by $N(x)$, that is,

$$\boxed{d'(x) = \alpha(G[N(x)]).} \tag{1}$$

Lemma 3. *Let G be a cograph. Then*

$$\boxed{\alpha'(G) = \max \{ \sum_{x \in W} d'(x) \mid W \text{ is an independent set in } G \}.} \tag{2}$$

Proof. Cographs are characterized by the fact that every induced subgraph has a twin. Let x be a false twin of a vertex y in G. Let A be a maximum independent set of edges in G. Let $A(x)$ and $A(y)$ be the sets of edges in A that are incident with x and y, respectively. Assume that $|A(x)| \geq |A(y)|$. Let $\Omega(x)$ be the set of end vertices in $N(x)$ of edges in $A(x)$. Then we may replace the set $A(y)$ with the set

$$\{ \{y, z\} \mid z \in \Omega(x) \}.$$

The cardinality of the new set is at least as large as $|A|$. Notice that, for any maximal independent set Q in G, either $\{x, y\} \subseteq Q$ or $\{x, y\} \cap Q = \varnothing$. By induction on the number of vertices in G, Equation (2) is valid.

Let x be a true twin of a vertex y in G. Let A be a maximum independent set of edges in G and let $A(x)$ and $A(y)$ be the sets of edges in A that are incident with x and y, respectively. If $\{x, y\} \in A$ then $A(x) = A(y) = \{\{x, y\}\}$.

Now assume that $\{x, y\} \notin A$. End vertices in $N(x)$ of edges in $A(x)$ and $A(y)$ are not adjacent nor do they coincide. Replace $A(x)$ with

$$\{ \{x, z\} \mid \{x, z\} \in A(x) \quad \text{or} \quad \{y, z\} \in A(y) \}$$

and set $A(y) = \varnothing$. Then the new set of edges is independent and has the same cardinality as A.

Let Q be an independent set in G. At most one of x and y is in Q. By induction on the number of vertices in G, the validity of Equation (2) is easily checked. □

We show that Equation (2) for a cograph can be computed in linear time in terms of the number of edges of the graph in the following theorem, whose proof is omitted due to lack of space.

Theorem 3. *When G is a cograph then $\alpha'(G)$ satisfies Equation* (2)*. This value can be computed in $O(n^2)$ time.*

Remark 2. For any graph G whose edge-clique graph is perfect the intersection number of G equals the fractional intersection number of G [23, Theorem 4.1]. It is easy to see that for cographs G, $K_e(G)$ is not necessarily perfect. For example, when G is the join of P_3 and C_4, then $K_e(G)$ contains a C_5 as an induced subgraph.[2]

3.2 Distance-Hereditary Graphs

A graph G is distance-hereditary if the distance between any two nonadjacent vertices, in any connected induced subgraph of G, is the same as their distance in the G [21]. Distance-hereditary graphs is a superclass of cographs. Bandelt and Mulder obtained the following characterization of distance-hereditary graphs.

Lemma 4 ([5]). *A graph is distance-hereditary if and only if every induced subgraph has an isolated vertex, a pendant vertex or a twin.*

The papers [5] and [21] also contain characterizations of distance-hereditary graphs in terms of forbidden induced subgraphs. We then show that Equation (2) for a distance-hereditary graph can be computed in polynomial time.

Theorem 4. *Let G be distance-hereditary. Then $\alpha'(G)$ satisfies Equation* (2)*. This value can be computed in polynomial time.*

Proof. The characterization of distance-hereditary graphs follows from lemma 4.

Consider an isolated vertex x in G. Then A is a maximum independent set of edges in G if and only if A is a maximum independent set of edges in the graph $G - x$. By induction, Equation (2) is valid for G.

Let x be a pendant vertex and let y be the unique neighbor of x in G. Since $\{x, y\}$ is not in any triangle, the edge $\{x, y\}$ is in any maximal independent set of edges in G. Therefore,

$$\alpha'(G) = 1 + \alpha'(G - x).$$

Let Q be an independent set which maximizes Equation (2) for $G - x$. If $y \in Q$ then $d'(y)$ goes up by one when adding the vertex x. If $y \notin Q$, then $Q \cup \{x\}$ is an independent set in G and $d'(x) = 1$.

By Lemma 3, Equation (2) is valid for true twins and false twins. Finally, this proves the theorem. □

[2] Let $P_3 = \{1, 2, 3\}$ and $C_4 = \{4, 5, 6, 7\}$. Then $K_e(G)$ contains an induced $C_5 = \{\{1, 2\}, \{3, 4\}, \{3, 7\}, \{4, 5\}, \{6, 7\}\}$.

3.3 P_4-Sparse Graphs

A graph is P_4-sparse if no set of five vertices induces more than one P_4 [22]. P_4-sparse graphs generalize the class of cographs and Jamison and Olariu characterize P_4-sparse graphs using spiders. Similar result for the class of P_4-sparse graphs is shown in Theorem 5. We omit its proof due to lack of space.

Theorem 5. *When G is P_4-sparse then $\alpha'(G)$ satisfies Equation (2). Furthermore, there exists a polynomial-time algorithm to compute a maximum independent set of edges in G.*

4 Algorithms for Planar Graphs

First, we use Baker's technique [4] to obtain a PTAS for a maximum independent set of edges in planar graphs in Theorem 6. We omit its proof due to lack of space. We leave open the question whether the edge-independent set problem for planar graphs is NP-complete.

Theorem 6. *There exists a polynomial-time approximation scheme for a maximum independent set of edges in planar graphs.*

4.1 Planar Graphs without Triangle Separator

Next, we show that the edge-independent set problem for a special class of planar graphs can be solved in polynomial time. A triangle separator (of a connected graph) is a triangle the removal of which disconnects the graph. We compute $\alpha'(G)$ for planar graphs G without triangle separator in the following theorem.

Theorem 7. *There exists an $O(n^{3/2})$ algorithm that computes a maximum independent set of edges in planar graphs without triangle separator.*

Proof. Let G be an embedding of a planar graph without triangle separator. We may assume that $G \not\simeq K_4$. In all the faces that have length more than three add a new vertex and make it adjacent to all vertices in the face. Let G' be this graph. Give the edges of G a weight 1 and give the new edges a weight 0.

Let H be the dual of G'. In H the weight of an edge is the weight of the edge in G' that it crosses. We claim that a solution is obtained by computing a maximum weight matching in H [14].

Let M^d be a matching in H. We may assume that M^d contains no edge of weight 0. Then every edge of M^d crosses an edge of G. We claim that this is an independent set M of edges in G. Since M^d is a matching no two triangular faces of G' incident with different edges of M^d coincide. Assume that two edges of M lie in a triangle T of G. Then T cannot be a face of G' since M^d is a matching. But then T is a separator which contradicts our assumption.

Let M be an independent set of edges in G and let M^d be the corresponding edges of H. Since M is an independent set of edges no two edges of M lie in a triangle. Assume that M^d is not a matching, and let f be a common face of G' of two edges in M^d. Then f is contained in a face of G which is not a triangle. But then the edges of M are the same. □

5 NP-Completeness for Graphs without Odd Wheels

A wheel W_n is a graph consisting of a cycle C_n and one additional vertex adjacent to all vertices in the cycle. The universal vertex of W_n is called the hub. It is unique unless $W_n = K_4$. The edges incident with the hub are called the spokes of the wheel. The cycle is called the rim of the wheel. A wheel is odd if the number of vertices in the cycle is odd.

Lakshmanan, Bujtás and Tuza investigate the class of graphs without odd wheels in [26]. They prove that Tuza's conjecture is true for this class of graphs (see also [20]). Notice that a graph G has no odd wheel if and only if every neighborhood in G induces a bipartite graph. It follows that $\omega(G) \leq 3$. Obviously, the class of graphs without odd wheels is closed under taking subgraphs.

Notice that, when G has no odd wheel then every neighborhood in $K_e(G)$ is either empty or a matching. Furthermore, it is easy to see that $K_e(G)$ contains no diamond (every edge is in exactly one triangle), no C_5 and no odd antihole.

For graphs G without odd wheels $K_e(G)$ coincides with the anti-Gallai graphs introduced by Le [28], since $\omega(G) \leq 3$. For general anti-Gallai graphs the computation of the clique number and chromatic number are NP-complete.

Let us mention that the recognition of anti-Gallai graphs is NP-complete. Even when each edge in G is in exactly one triangle, the problem to decide if G is an anti-Gallai graph is NP-complete [3, Corollary 5.2]. The recognition of edge-clique graphs of graphs without odd wheels is, as far as we know, open. Let us also mention that the edge-clique graphs of graphs without odd wheels are clique graphs [7]. The recognition of clique graphs of general graphs is NP-complete [2]. We show below that the edge-independent set problem for graphs without odd wheels in NP-complete.

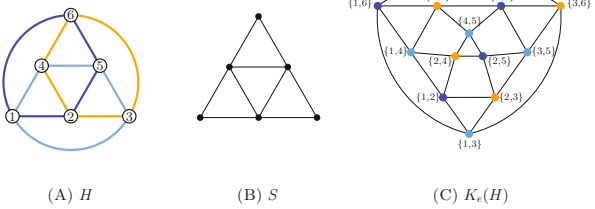
Theorem 8. *The computation of $\alpha'(G)$ is NP-complete for graphs G without odd wheels.*

Proof. It is clear that the problem is in NP. Then we reduce 3-SAT to the vertex cover problem in edge-clique graphs of graphs without odd wheels.
Let $H \simeq cp(3)$, i.e., the line graph of K_4. See Fig. 1(A). Let S be a 3-sun as depicted in Fig. 1(B). The graph H is obtained from S by adding three edges between pairs of vertices of degree two in S.[3] Call the three vertices of degree four in S, the 'inner triangle' of H and call the remaining three vertices of H the 'outer triangle.'

Notice that H has 3 maximum independent sets of edges. Each maximum independent set of edges is an induced C_4 consisting of one edge from the inner triangle, one edge from the outer triangle, and two edges between the two triangles. The three independent sets partition the edges of H.

The six edges of H between the inner and outer triangle form a 6-cycle in $K_e(H)$. Let F denote this set of edges in H.

[3] In [27, Theorem 14] the authors prove that every maximal clique in $K_e(G)$ contains a simplicial vertex if and only if G does not contain, as an induced subgraph, K_4 nor a 3-sun with 0, 1, 2 or 3 edges connecting the vertices of degree two.



(A) H (B) S (C) $K_e(H)$

Fig. 1. This figure shows H, S and $K_e(H)$. In H, vertices 2, 4, 5 induce the inner triangle and the remaining three vertices induce the outer triangle. The three colors for the edges of H indicate the partition of three maximum independent sets of edges.

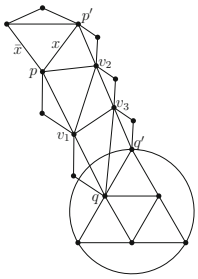


Fig. 2. A link gadget between a variable gadget and a clause gadget

For each clause $(x_i \vee x_j \vee x_k)$ take one copy of H. Take an independent set of three edges contained in F and label these with x_i, x_j and x_k.
For each variable x take a triangle. Label one edge of the triangle with the literal x and one other edge of the triangle with its negation $\bar{x}$. Then add a simplicial vertex to the unlabeled edge of the triangle.

Construct links between variable gadgets and clause gadgets as follows. Let $(x_i \vee x_j \vee x_k)$ be a clause. There are four steps. First add a 2-chain pv_1q from an endpoint p of the edge labeled x_i in the variable gadget to one endpoint q of the edge x_i in the clause gadget. See Fig. 2. Similarly, add a 3-chain $p'v_2v_3q'$ between the other two endpoints, where p' is in the variable gadget and q' is in the clause gadget. Then add four edges pv_2, v_2v_1, v_1v_3 and v_3q to let the rectangle $pqq'p'$ become a triangle strip with five triangles. Finally, add a simplicial vertex to each of the edges in the 2-chain and the 3-chain. We construct links for the other two literals in the clause in the same manner.

Let G be the graph constructed in this manner. Consider the triangles T which are the edge-clique graphs of the triangles in G containing the newly-added simplicial vertices. Notice that simplicial vertices of a graph may be removed without changing the complexity of the vertex cover problem and so the vertices of T can be removed without changing the complexity. Let K be the graph obtained from $K_e(G)$ by removing the vertices of T.

Let L be the number of variables, let M be the number of clauses in the 3-SAT formula. Assume that there is a satisfying assignment. Then choose the vertices in K corresponding to literals that are TRUE in the vertex cover. The variable gadgets need L vertices and the links need $6M$ vertices in the vertex cover. Since this assignment is satisfying, we need at most $8M$ vertices to cover the remaining edges in the clause structures, since the outgoing edge from each literal which is TRUE is covered. Thus there is a vertex cover of $K_e(G)$ with $L + 14M$ vertices.

Assume that $K_e(G)$ has a vertex cover with $L + 14M$ vertices. At least L vertices in K are covering the edges in the variable gadgets and at least $6M$ vertices in K are covering the edges in the links. The other $8M$ vertices of K are

covering the edges in the clause gadgets. Take the literals of the variable gadgets that are in the vertex cover as an assignment for the formula. Each clause gadget must have one literal vertex of which the outgoing edge is covered. Therefore, the assignment is satisfying.

This proves the theorem. □

6 Concluding Remarks

Cygan et al [12] show that, under the assumption of the exponential time hypothesis, there is no polynomial-time algorithm which reduces the parameterized problem $(\theta_e(G), k)$ to a kernel of size bounded by $2^{o(k)}$. In their proof the authors make use of the fact that $\theta_e(cp(2^\ell))$ is a "hard instance for the edge clique cover problem, at least from the point of view of the currently known algorithms." (Quotation from [12].) Note that, in contrast, the parameterized edge-clique partition problem can be reduced to a kernel with at most k^2 vertices [29]. (Mujuni and Rosamond also mention that the edge-clique cover problem probably has no polynomial kernel.) These observations lead us to investigate edge-clique graphs of cocktail parties in Section 2.

Let K_n^m denote the complete multi-partite graph with m partite sets each having n vertices. Obviously, K_n^m is a cograph with $n \cdot m$ vertices.

Theorem 9 ([31]). *Assume that*

$$3 \leq m \leq n + 1.$$

Then $\theta_e(K_n^m) = n^2$ if and only if there exists a collection of at least $m - 2$ pairwise orthogonal Latin squares of order n.

Notice that, if there exists an edge-clique cover of K_n^m with n^2 cliques, then these cliques are mutually edge-disjoint. Finding the maximal number of mutually orthogonal Latin squares of order n is a renowned open problem. The problem has a wide field of applications, eg in combinatorics, designs of experiments, group theory and quantum informatics.

Unless n is a prime power, the maximal number of MOLS is known for only a few orders. We briefly mention a few results. Let $f(n)$ denote the maximal number of MOLS of order n. The well-known 'Euler-spoiler' shows that $f(n) = 1$ only for $n = 2$ and $n = 6$. Also, $f(n) \leq n - 1$ for all $n > 1$, and Chowla, Erdös and Straus [10] show that

$$\lim_{n \to \infty} f(n) = \infty.$$

Define

$$n_r = \max \{ n \mid f(n) < r \}.$$

A lowerbound for the speed at which $f(n)$ grows was obtained by Wilson, who showed that $n_r < r^{17}$ when r is sufficiently large [35]. Better bounds for n_r, for some specific values of r, were obtained by various authors (see eg [6]).

See eg [25] for some recent computational attempts to find orthogonal Latin squares. The problem seems extremely hard, both from a combinatorial and from a computational point of view [24]. Despite many efforts, the existence of three pairwise orthogonal Latin squares of order 10 is, as far as we know, still unclear.

Finally, the observations mentioned above lead us to conjecture that the edge-clique cover problem is NP-complete for cographs.

References

1. Albertson, M., Collins, K.: Duality and perfection for edges in cliques. Journal of Combinatorial Theory, Series B 36, 298–309 (1984)
2. Alcón, L., Faria, L., de Figueiredo, C., Gutierrez, M.: The complexity of clique graph recognition. Theoretical Computer Science 410, 2072–2083 (2009)
3. Anand, P., Escuadro, H., Gera, R., Hartke, S., Stolee, D.: On the hardness of recognizing triangular line graphs. Discrete Mathematics 312, 2627–2638 (2012)
4. Baker, B.: Approximation algorithms for NP-complete problems on planar graphs. Journal of the ACM 41, 153–180 (1994)
5. Bandelt, H., Mulder, H.: Distance-hereditary graphs. Journal of Combinatorial Theory, Series B 41, 182–208 (1986)
6. Brouwer, A., Rees, G.: More mutually orthogonal Latin squares. Discrete Mathematics 39, 181–263 (1982)
7. Cerioli, M.: Clique graphs and edge-clique graphs. Electronic Notes in Discrete Mathematics 13, 34–37 (2003)
8. Cerioli, M., Szwarcfiter, J.: A characterization of edge clique graphs. Ars Combinatorica 60, 287–292 (2001)
9. Cerioli, M., Szwarcfiter, J.: Edge clique graphs and some classes of chordal graphs. Discrete Mathematics 242, 31–39 (2002)
10. Chowla, S., Erdös, P., Straus, E.: On the maximal number of pairwise orthogonal Latin squares of a given order. Canadian Journal of Mathematics 12, 204–208 (1960)
11. Corneil, D., Perl, Y., Stewart, L.: A linear recognition algorithm for cographs. SIAM Journal on Computing 14, 926–934 (1985)
12. Cygan, M., Pilipczuk, M., Pilipczuk, M.: Known algorithms for edge clique cover are probably optimal. In: Proceedings SODA 2013, ACM-SIAM, pp. 1044–1053 (2013)
13. Dvořák, Z., Král, D.: Classes of graphs with small rank decompositions are χ-bounded. European Journal of Combinatorics 33, 679–683 (2012)
14. Edmonds, J.: Paths, trees, and flowers. Canadian Journal of Mathematics 17, 449–467 (1965)
15. Ganian, R., Hliněný, P.: Better polynomial algorithms on graphs of bounded rank-width. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 266–277. Springer, Heidelberg (2009)
16. Gao, J., Kloks, T., Poon, S.-H.: Triangle-partitioning edges of planar graphs, toroidal graphs and k-planar graphs. In: Ghosh, S.K., Tokuyama, T. (eds.) WALCOM 2013. LNCS, vol. 7748, pp. 194–205. Springer, Heidelberg (2013)
17. Gargano, L., Körner, J., Vaccaro, U.: Sperner capacities. Graphs and Combinatorics 9, 31–46 (1993)
18. Gregory, D.A., Pullman, N.J.: On a clique covering problem of Orlin. Discrete Mathematics 41, 97–99 (1982)

19. Gyárfás, A.: A simple lower bound on edge covering by cliques. Discrete Mathematics 85, 103–104 (1990)
20. Haxell, P., Kostoschka, A., Thomassé, S.: A stability theorem on fractional covering of triangles by edges. European Journal of Combinatorics 33, 799–806 (2012)
21. Howork, E.: A characterization of distance-hereditary graphs. The Quarterly Journal of Mathematics 28, 417–420 (1977)
22. Jamison, B., Olariu, S.: A unique tree representation for P_4-sparse graphs. Discrete Applied Mathematics 35, 115–129 (1992)
23. Kong, J., Wu, Y.: On economical set representations of graphs. Discrete Mathematics and Theoretical Computer Science 11, 71–96 (2009)
24. Körner, J.: Intersection number and capacities of graphs. Discrete Mathematics 142, 169–184 (1995)
25. Ma, F., Zhang, J.: Finding orthogonal Latin squares using finite model searching tools. Science China Information Sciences 56, 1–9 (2013)
26. Lakshmanan, S., Bujtás, C., Tuza, Z.: Small edge sets meeting all triangles of a graph. Graphs and Combinatorics 28, 381–392 (2012)
27. Lakshmanan, S., Vijayakumar, A.: Clique irreducibility of some iterative classes of graphs. Discussiones Mathematicae Graph Theory 28, 307–321 (2008)
28. Le, V.B.: Gallai graphs and anti-Gallai graphs. Discrete Mathematics 159, 179–189 (1996)
29. Mujuni, E., Rosamond, F.: Parameterized complexity of the clique partition problem. In: Harland, J., Manyem, P. (eds.) Proceedings CATS 2008. ACS, CRPIT series, vol. 77, pp. 75–78 (2008)
30. Oum, S.: Graphs of bounded rankwidth. PhD thesis, Princeton University (2005)
31. Park, B., Kim, S., Sano, Y.: The competition numbers of complete multipartite graphs and mutually orthogonal Latin squares. Discrete Mathematics 309, 6464–6469 (2009)
32. Prisner, E.: Graph dynamics. Pitman Research Notes in Mathematics Series. Longman, Essex (1995)
33. Raychaudhuri, A.: Intersection number and edge clique graphs of chordal and strongly chordal graphs. Congressus Numerantium 67, 197–204 (1988)
34. Raychaudhuri, A.: Edge clique graphs of some important classes of graphs. Ars Combinatoria 32, 269–278 (1991)
35. Wilson, R.: Concerning the number of mutually orthogonal Latin squares. Discrete Mathematics 9, 181–198 (1974)

On the Complexity of Approximate Sum of Sorted List*

Bin Fu

Department of Computer Science
University of Texas-Pan American, Edinburg, TX 78539, USA
bfu@utpa.edu

Abstract. We consider the complexity for computing the approximate sum $a_1 + a_2 + \cdots + a_n$ of a sorted list of numbers $a_1 \leq a_2 \leq \cdots \leq a_n$. We show an algorithm that computes an $(1+\epsilon)$-approximation for the sum of a sorted list of nonnegative numbers in an $O(\frac{1}{\epsilon}\min(\log n, \log(\frac{x_{max}}{x_{min}})) \cdot (\log\frac{1}{\epsilon} + \log\log n))$ time, where x_{max} and x_{min} are the largest and the least positive elements of the input list, respectively. We prove a lower bound $\Omega(\min(\log n, \log(\frac{x_{max}}{x_{min}}))$ time for every $O(1)$-approximation algorithm for the sum of a sorted list of nonnegative elements. We also show that there is no sublinear time approximation algorithm for the sum of a sorted list that contains at least one negative number.

1 Introduction

Computing the sum of a list of numbers is a classical problem that is often found inside the high school textbooks. There is a famous story about Karl Friedrich Gauss who computed $1+2+\cdots+100$ via rearranging these terms into $(1+100)+(2+99)+...+(50+51) = 50 \times 101$, when he was seven years old, attending elementary school. Such a method is considered an efficient algorithm for computing a class of lists of increasing numbers. Computing the sum of a list of elements has many applications, and is ubiquitous in software design. In the classical mathematics, many functions can be approximated by the sum of simple functions via Taylor expansion. This kind of approximation theories is in the core area of mathematical analysis. In this article we consider if there is an efficient way to compute the sum of a general list of nonnegative numbers with nondecreasing order.

Let ϵ be a real number at least 0. Real number s is an $(1+\epsilon)$-approximation for the sum problem $a_1, a_2, \cdots, a_n$ if $\frac{\sum_{i=1}^{n} a_i}{1+\epsilon} \leq s \leq (1+\epsilon)\sum_{i=1}^{n} a_i$. Approximate sum problem was studied in the randomized computation model. Every $O(1)$-approximation algorithm with uniform random sampling requires $\Omega(n)$ time in the worst case if the list of numbers in $[0,1]$ is not sorted. Using $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ random samples, one can compute the $(1+\epsilon)$-approximation for the mean, or decide

* This research is supported in part by NSF HRD-1137764 and NSF Early Career Award 0845376.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 284–293, 2013.

if it is at most δ for a list numbers in $[0, 1]$ [9]. Canetti, Even, and Goldreich [3] showed that the sample size is tight. Motwani, Panigrahy, and Xu [14] showed an $O(\sqrt{n})$ time approximation scheme for computing the sum of n nonnegative elements. There is a long history of research for the accuracy of summation of floating point numbers (for examples, see [10, 2, 1, 4–6, 8, 11–13, 15, 16]). The efforts were mainly spent on finding algorithms with small rounding errors.

We investigate the complexity for computing the approximate sum of a sorted list. When we have a large number of data items and need to compute the sum, an efficient approximation algorithm becomes important. Har-Peled developed an coreset approach for a more general problem. The method used in his paper implies an $O(\frac{\log n}{\epsilon})$ time approximation algorithm for the approximate sum of sorted nonnegative numbers [7]. The coreset is a subset of numbers selected from a sorted input list, and their positions only depends on the size n of the list, and independent of the numbers. The coreset of a list of n sorted nonnegative numbers has a size $\Omega(\log n)$. This requires the algorithm time to be also $\Omega(\log n)$ under all cases.

We show an algorithm that gives an $(1 + \epsilon)$-approximation for the sum of a list of sorted nonnegative elements in $O(\frac{1}{\epsilon} \min(\log n, \log(\frac{x_{max}}{x_{min}})) \cdot (\log \frac{1}{\epsilon} + \log\log n))$ time, where x_{max} and x_{min} are the largest and the least positive elements of the input list, respectively. This algorithm has an incomparable complexity with Har-Peled's algorithm. Our algorithm is of sub-logarithm complexity when $\frac{x_{max}}{x_{min}} \leq n^{\frac{1}{(\log\log n)^{1+a}}}$ for any fixed $a > 0$. The algorithm is based on a different method, which is a quadratic region search algorithm, from the coreset construction used in [7].

We also prove a lower bound $\Omega(\min(\log n, \log(\frac{x_{max}}{x_{min}}))$ for this problem. We first derive an $O(\log\log n)$ time approximation algorithm that finds an approximate region of the list for holding the items of size at least a threshold b. Our approximate sum algorithm is derived with it as a submodule. We also show an $\Omega(\log\log n)$ lower bound for approximate region algorithms for the sum of a sorted list with only nonnegative elements.

In Section 2, we present an algorithm that computes $(1 + \epsilon)$-approximation for the sum of a sorted list of nonnegative numbers In Section 3, we present lower bounds related to the sum of sorted list. In Section **??**, we show the experimental results for the implementation of our algorithm in Section 2.

2 Algorithm for Approximate Sum of Sorted List

In this section, we show a deterministic algorithm for the sorted elements. We first show an approximation to find an approximate region of a sorted list with elements of size at least threshold b.

A crucial part of our approximate algorithm for the sum of sorted list is to find an approximate region with elements of size at least a threshold b. We develop a method that is much faster than binary search and it takes $O(\log \frac{1}{\delta} + \log\log n)$ time to find the approximate region. We first apply the square function to expand the region and use the square root function to narrow down to a region that only

has $(1+\delta)$ factor difference with the exact region. The parameter δ determines the accuracy of approximation.

Definition 1. *For $i \le j$, let $|[i,j]|$ be the number of integers in the interval $[i,j]$.*

If both i and j are integers with $i \le j$, we have $|[i,j]| = j - i + 1$.

Definition 2. *A list X of n numbers is represented by an array $X[1,n]$, which has n numbers $X[1], X[2], \cdots, X[n]$. For integers $i \le j$, let $X[i,j]$ be the sublist that contains elements $X[i], X[i+1], \cdots, X[j]$. For an interval $R = [i,j]$, denote $X[R]$ to be $X[i,j]$.*

Definition 3. *For a sorted list $X[1,n]$ with nonnegative elements by nondecreasing order and a threshold b, the b-region is an interval $[n',n]$ such that $X[n',n]$ are the numbers at least b in $X[1,n]$. An $(1+\delta)$-approximation for the b-region is a region $R = [s,n]$ such that at least $\frac{|R|}{1+\delta}$ numbers in $X[s,n]$ are at least b, and $[s,n]$ contains all every position j with $X[j] \ge b$.*

2.1 Approximate Region

The approximation algorithm for finding an approximate b-region to contain the elements at least a threshold b has two loops. The first loop searches the region by increasing the parameter m via the square function. When the region is larger than the exact region, the second loop is entered. It converges to the approximate region with a factor that goes down by a square root each cycle. Using the combination of the square and square root functions makes our algorithm much faster than the binary search.

In order to simplify the description of the algorithm Approximate-Region(.), we assume $X[i] = -\infty$ for every $i \le 0$. It can save the space for the boundary checking when accessing the list X. The description of the algorithm is mainly based on the consideration for its proof of correctness. For a real number a, denote $\lfloor a \rfloor$ to be the largest integer at most a, and $\lceil a \rceil$ to be the least integer at least a.

Algorithm Approximate-Region(X, b, δ, n)

Input: $X[1,n]$ is a sorted list of n numbers by nondecreasing order; n is the size of $X[1,n]$; b is a threshold in $(0,+\infty)$; and δ is a parameter in $(0,+\infty)$.

```
1.    if (X[n] < b), return ∅;
2.    if (X[n − 1] < b), return [n, n];
3.    if (X[1] ≥ b), return [1, n];
4.    let m := 2;
5.    while (X[n − m^2 + 1] ≥ b) {
6.          let m := m^2;
7.    };
8.    let i := 1;
9.    let m_1 := m;
```

10. let $r_1 := m$;
11. while $(m_i \geq 1 + \delta)$ {
12. let $m_{i+1} := \sqrt{m_i}$;
13. if $(X[n - \lfloor m_{i+1} r_i \rfloor + 1] \geq b)$, then let $r_{i+1} := m_{i+1} r_i$;
14. else $r_{i+1} := r_i$;
15. let $i := i + 1$;
16. };
17. return $[n - \lfloor m_i r_i \rfloor + 1, n]$;

End of Algorithm

Lemma 1. *Let δ be a parameter in $(0, 1)$. Given an element b, and a list A of sorted n elements, Algorithm Approximate-Region(.) finds an $(1+\delta)$-approximate b-region in $O((\log \frac{1}{\delta}) + (\log \log n))$ time.*

Proof. After the first phase (lines 1 to 7) of the algorithm, we obtain number m such that

$$X[n - m + 1] \geq b, \qquad \text{and} \tag{1}$$
$$X[n - m^2 + 1] < b. \tag{2}$$

As we already assume $X[i] = -\infty$ for every $i \leq 0$, there is no boundary problem for assessing the input list. The variable m is an integer in the first phase. Thus, the boundary point for the region with numbers at least the threshold b is in $[n - m^2 + 1, n - m + 1]$. The variable m can be expressed as 2^{2^k} for some integer $k \geq 0$ after executing k cycles in the first phase. Thus, the first phase takes $O(\log \log n)$ time because m is increased to m^2 at each cycle of the first while loop, and $2^{2^k} \geq n$ for $k \geq \log \log n$.

In the second phase (lines 8 to 17) of the algorithm, we can prove that $X[n - \lfloor r_i \rfloor + 1] \geq b$ and $X[n - \lfloor m_i r_i \rfloor + 1] < b$ at the end of every cycle (right after executing the statement at line 15) of the second loop (lines 11 to 16). Thus, the boundary point for the region with elements at the threshold b is in $[n - \lfloor m_i r_i \rfloor + 1, n - \lfloor r_i \rfloor + 1]$. The variable m_i is not an integer after $m_i < 2$ in the algorithm. It can be verified via a simple induction. It is true before entering the second loop (lines 11 to 16) by inequalities (1) and (2). Assume that at the end of cycle i,

$$X[n - \lfloor r_i \rfloor + 1] \geq b; \quad \text{and} \tag{3}$$
$$X[n - \lfloor m_i r_i \rfloor + 1] < b. \tag{4}$$

Let us consider cycle $i + 1$ at the second loop. Let $m_{i+1} = \sqrt{m_i}$.

1. Case 1: $X[n - \lfloor m_{i+1} r_i \rfloor + 1] \geq b$. Let $r_{i+1} = m_{i+1} r_i$ according to line 13 in the algorithm. Then $X[n - \lfloor r_{i+1} \rfloor + 1] = X[n - \lfloor m_{i+1} r_i \rfloor + 1] \geq b$. By inequality (4) in the hypothesis, $X[n - \lfloor m_{i+1} r_{i+1} \rfloor + 1] = X[n - \lfloor \sqrt{m_i} \sqrt{m_i} r_i \rfloor + 1] = X[n - \lfloor m_i r_i \rfloor + 1] < b$.

2. Case 2: $X[n - \lfloor m_{i+1}r_i \rfloor + 1] < b$. Let $r_{i+1} = r_i$ according to line 14 the algorithm. We have $X[n-\lfloor r_{i+1} \rfloor+1] = X[n-\lfloor r_i \rfloor+1] \geq b$ by inequality (3) in the hypothesis. By inequality (4) in the hypothesis, $X[n-\lfloor m_{i+1}r_{i+1} \rfloor+1] = X[n - \lfloor m_{i+1}r_i \rfloor + 1] < b$ by the condition of this case.

Therefore, $X[n - \lfloor r_{i+1} \rfloor + 1] \geq b$ and $X[n - \lfloor m_{i+1}r_{i+1} \rfloor + 1] < b$ at the end of cycle $i + 1$ of the second while loop.

Every number in $X[n - r_i + 1, n]$, which has r_i entries, is at least b, and $X[n - m_i r_i + 1, n]$ has $m_i r_i$ entries and $m_i \leq 1 + \delta$ at the end of the algorithm. Thus, the interval $[n - m_i r_i + 1, n]$ returned by the algorithm is an $(1 + \delta)$-approximation for the b-region.

It takes $O(\log \log n)$ steps for converting m to be at most 2, and additional $\log \frac{1}{\delta}$ steps to make m to be at most $1 + \delta$. When $m_i < 1 + \delta$, we stop the loop, and output an $(1+\delta)$-approximation. This step takes at most $O(\log \frac{1}{\delta} + \log \log n)$ time since m_i is assigned to $\sqrt{m_i}$ at each cycle of the second loop. This proves Lemma 1.

After the first loop of the algorithm Approximate-Region(.), the number m is always of the format 2^{2^k} for some integer k. In the second loop of the algorithm Approximate-Region(.), the number m is always of the format 2^{2^k} when m is at least 2. Computing its square root is to convert 2^{2^k} to $2^{2^{k-1}}$, where k is an integer. Since $(1 + \frac{1}{2^i}) \cdot (1 + \frac{1}{2^i}) > (1 + \frac{1}{2^{i-1}})$, we have that $(1 + \frac{1}{2^i})$ is larger than the square root of $(1 + \frac{1}{2^{i-1}})$. We may let variable m_i go down by following the sequence $\{(1 + \frac{1}{2^i})\}_{i=1}^{\infty}$ after $m_i \leq 2$. In other words, let $g(.)$ be an approximate square root function such that $g(1 + \frac{1}{2^i}) = 1 + \frac{1}{2^{i+1}}$ for computing the square root after $m \leq 2$ in the algorithm. It has the property $g(m) \cdot g(m) \geq m$. The assignment $m_{i+1} = \sqrt{m_i}$ can be replaced by $m_{i+1} = g(m_i)$ in the algorithm. It can simplify the algorithm by removing the computation of square root while the computational complexity is of the same order.

2.2 Approximate Sum

We present an algorithm to compute the approximate sum of a list of sorted nonnegative elements. It calls the module for the approximate region, which is described in Section 2.1.

The algorithm for the approximate sum of a sorted list X of n nonnegative numbers generates a series of disjoint intervals $R_1 = [r_1, r'_1], \cdots, R_t = [r_t, r'_t]$, and a series of thresholds $b_1, \cdots, b_t$ such that each R_i is an $(1 + \delta)$-approximate b_i-region in $X[1, r'_i]$, $r'_1 = n$, $r'_{i+1} = r_i - 1$, and $b_{i+1} \leq \frac{b_i}{1+\delta}$, where $\delta = \frac{3\epsilon}{4}$ and $1 + \epsilon$ is the accuracy for approximation. The sum of numbers in $X[R_i]$ is approximated by $|R_i|b_i$. As the list $b_1 > b_2 > \cdots > b_t$ decreases exponentially, we can show that $t = O(\frac{1}{\epsilon} \log n)$. The approximate sum for the input list is $\sum_{i=1}^{t} |R_i|b_i$. We give a formal description of the algorithm and its proof below.

Algorithm Approximate-Sum(X, ϵ, n)

Input: $X[1, n]$ is a sorted list of nonnegative numbers (by nondecreasing order) and n is the size of $X[1, n]$, and ϵ is a parameter in $(0, 1)$ for the accuracy of approximation.

1. if $(X(n) = 0)$, return 0;
2. let $\delta := \frac{3\epsilon}{4}$;
3. let $r'_1 := n$;
4. let $s := 0$;
5. let $i := 1$;
6. let $b_1 := \frac{X[n]}{1+\delta}$;
7. while $(b_i \geq \frac{\delta X[n]}{3n})$ {
8. let $R_i :=$Approximate-Region(X, b_i, δ, r'_i);
9. let $r'_{i+1} := r_i - 1$ for $R_i = [r_i, r'_i]$;
10. let $b_{i+1} := \frac{X[r'_{i+1}]}{1+\delta}$;
11. let $s_i := |[r_i, r'_i]| \cdot b_i$;
12. let $s := s + s_i$;
13. let $i := i + 1$;
14. };
15. return s;

End of Algorithm

Theorem 1. *Let ϵ be a positive parameter. Then there is an $O(\frac{1}{\epsilon}\min(\log n, \log(\frac{x_{max}}{x_{min}})) \cdot (\log\frac{1}{\epsilon} + \log\log n))$ time algorithm to compute $(1+\epsilon)$-approximation for the sum of sorted list of nonnegative numbers, where x_{max} and x_{min} are the largest and the least positive elements of the input list, respectively.*

Proof. Assume that there are t cycles executed in the while loop of the algorithm Approximate-Sum(.). Let regions $R_1, R_2, \cdots, R_t$ be generated. In the first cycle of the loop, the algorithm finds a region $R_1 = [r_1, n]$ of the elements of size at least $\frac{X[n]}{1+\delta}$. In the second cycle of the loop, the algorithm finds region $R_2 = [r_2, r_1 - 1]$ for the elements of size at least $\frac{X[r_1-1]}{1+\delta}$. In the i-th cycle of the loop, it finds a region $R_i = [r_i, r_{i-1} - 1]$ of elements of size at least $\frac{X[r_{i-1}-1]}{1+\delta}$. By the algorithm, we have

$$j \in R_1 \cup R_2 \cup \cdots \cup R_t \quad \text{for every } j \text{ with } X[j] \geq \frac{\delta X[n]}{3n}. \tag{5}$$

Since each R_i is an $(1+\delta)$-approximation of $\frac{X[r_{i-1}-1]}{1+\delta}$-region in $X[1, r_{i-1} - 1]$, $X[R_i]$ contains at least $\frac{|R_i|}{1+\delta}$ entries of size at least $\frac{X[r_{i-1}-1]}{1+\delta}$ in $X[1, r_{i-1} - 1]$, R_i also contains every entry of size at least $\frac{X[r_{i-1}-1]}{1+\delta}$ in $X[1, r_{i-1} - 1]$. Thus,

$$\frac{s_i}{1+\delta} = \frac{|R_i|}{1+\delta} \cdot \frac{X[r_{i-1} - 1]}{1+\delta} \leq \sum_{j \in R_i} X[j] \leq |R_i| X[r_{i-1} - 1] = (1+\delta)s_i.$$

Thus,

$$\frac{s_i}{1+\delta} \leq \sum_{j \in R_i} X[j] \leq (1+\delta)s_i.$$

We have

$$\frac{1}{1+\delta} \sum_{j \in R_i} X[j] \leq s_i \leq (1+\delta) \sum_{j \in R_i} X[j]. \tag{6}$$

Thus, s_i is an $(1+\delta)$-approximation for $\sum_{j \in R_i} X[j]$. We also have $\sum_{X[i] < \frac{\delta X[n]}{3n}} X[i] < \frac{\delta X[n]}{3}$ since $X[1,n]$ has only n numbers in total. Therefore, we have the following inequalities:

$$\sum_{X[i] \geq \frac{\delta X[n]}{3n}} X[i] = \sum_{i=1}^{n} X[i] - \sum_{X[i] < \frac{\delta X[n]}{3n}} X[i] \tag{7}$$

$$\geq \sum_{i=1}^{n} X[i] - \frac{\delta}{3} \sum_{i=1}^{n} X[i] \tag{8}$$

$$= (1 - \frac{\delta}{3}) \sum_{i=1}^{n} X[i]. \tag{9}$$

We have the inequalities:

$$s = \sum_{i=1}^{t} s_i \tag{10}$$

$$\geq \frac{1}{1+\delta} \sum_{X[i] \geq \frac{\delta X[n]}{3n}} X[i] \quad \text{(by inequality (6)))} \tag{11}$$

$$\geq \frac{(1-\frac{\delta}{3})}{1+\delta} \sum_{i=1}^{n} X[i] \quad \text{(by inequality (9))} \tag{12}$$

$$= \frac{1}{\frac{1+\delta}{1-\frac{\delta}{3}}} \sum_{i=1}^{n} X[i] \tag{13}$$

$$= \frac{1}{1 + \frac{\frac{4\delta}{3}}{1-\frac{\delta}{3}}} \sum_{i=1}^{n} X[i] \tag{14}$$

$$\geq \frac{1}{1+\frac{4\delta}{3}} \sum_{i=1}^{n} X[i] \tag{15}$$

$$= \frac{1}{1+\epsilon} \sum_{i=1}^{n} X[i]. \tag{16}$$

As $R_1, R_2, \cdots$ are disjoint each other, we also have the following inequalities:

$$s = \sum_{i=1}^{t} s_i \tag{17}$$

$$\leq \sum_{i=1}^{t}(1+\delta)\sum_{j\in R_i} X[j] \qquad \text{(by inequality (6))} \tag{18}$$

$$\leq (1+\delta)\sum_{j=1}^{n} X[j] \tag{19}$$

$$\leq (1+\epsilon)\sum_{j=1}^{n} X[j]. \tag{20}$$

Therefore, the output s returned by the algorithm is an $(1+\epsilon)$-approximation for the sum $\sum_{i=1}^{n} X[i]$. By Lemma 1, each cycle in the while loop of the algorithm takes $O((\log\frac{1}{\delta}+\log\log n))$ time for generating R_i. For the descending chain $r'_1 > r'_2 > \cdots > r'_t$ with $X[r'_i] \leq \frac{X[r'_{i+1}]}{1+\delta}$ and $b_i = X[r'_i] \geq \frac{\delta X[n]}{3n}$ for each i, we have that the number of cycles t is at most $O(\frac{1}{\delta}\log n)$. This is because $X[r'_t] \leq \frac{x_{max}}{(1+\delta)^t} \leq \frac{\delta X[n]}{3n}$ for some $t = O(\frac{1}{\delta}\log n)$. Similarly, the number of cycles t is at most $O(\frac{1}{\delta}\log(\frac{x_{max}}{x_{min}}))$ because $X[r'_t] \leq \frac{x_{max}}{(1+\delta)^t} \leq x_{min}$ for some $t = O(\frac{1}{\delta}\log(\frac{x_{max}}{x_{min}}))$.

Therefore, there are most $t = O(\frac{1}{\delta}\min(\log n, \log\frac{x_{max}}{x_{min}}))$ cycles in the while loop of the algorithm. Therefore, the total time is $O(\frac{1}{\delta}\min(\log n, \log(\frac{x_{max}}{x_{min}}))(\log\frac{1}{\delta}+\log\log n)) = O(\frac{1}{\epsilon}\min(\log n, \log(\frac{x_{max}}{x_{min}}))(\log\frac{1}{\epsilon}+\log\log n))$. This proves Theorem 1.

3 Lower Bounds

In this section, we show several lower bounds about approximation for the sum of sorted list. The $\Omega(\min(\log n, \log(\frac{x_{max}}{x_{min}}))$ lower bound is based on the general computation model for the sum problem. The lower bound $\Omega(\log\log n))$ for finding an approximate b-region shows that upper bound is optimal if using the method developed in Section 2. We also show that there is no sublinear time algorithm if the input list contains one negative element.

3.1 Lower Bound for Computing Approximate Sum

In this section, we show a lower bound for the general computation model, which almost matches the upper bound of our algorithm. This indicates the algorithm in Section 2 can be improved by at most $O(\log\log n)$ factor.

The lower bound is proved by a contradiction method. In the proof of the lower bound, two lists L_1 and L_2 are constructed. For an algorithm with $o(\log n)$ queries, the two lists will have the same answers to all queries. Thus, the approximation outputs for the two inputs L_1 and L_2 are the same. We let the gap of the sums from the two lists be large enough to make them impossible to share the same constant factor approximation.

Theorem 2. *For every positive constant $d > 1$, every d-approximation algorithm for the sum of a sorted list of nonnegative numbers needs at least*

$\Omega(\min(\log n, \log \frac{x_{max}}{x_{min}}))$ *(adaptive) queries to the list, where* γ *is an arbitrary small constant in* $(0, 1)$*, where* x_{max} *and* x_{min} *are the largest and the least positive elements of the input list, respectively..*

3.2 Lower Bound for Computing Approximate Region

We give an $\Omega(\log \log n)$ lower bound for the deterministic approximation scheme for a b-region in a sorted input list of nonnegative numbers. The method is that if there is an algorithm with $o(\log \log n)$ queries, two sorted lists L_1 and L_2 of $0, 1$ numbers are constructed. They reply the same answer the each the query from the algorithm, but their sums have large difference. This lower bound shows that it is impossible to use the method of Section 2, which iteratively finds approximate regions via a top down approach, to get a better upper bound for the approximate sum problem.

Definition 4. *For a sorted list* $X[1, n]$ *with* $0, 1$ *numbers by nondecreasing order, an* d*-approximate* 1*-region is a region* $R = [s, n]$*, which contains the last position* n *of* $X[1, n]$*, such that at least* $\frac{|R|}{d}$ *numbers in* $X[s, n]$ *are* 1*, and* $X[s, n]$ *contains all the positions* j *with* $X[j] = 1$*, where* $|R|$ *is the number of integers* i *in* R*.*

Theorem 3. *For any parameter* $d > 1$*, every deterministic algorithm must make at least* $\log \log n - \log \log(d + 1)$ *adaptive queries to a sorted input list for the* d*-approximate* 1*-region problem.*

Corollary 1. *For any constant* $\epsilon \in (0, 1)$*, every deterministic* $O(1)$*-approximation algorithm for* 1*-region problem must make at least* $(1 - \epsilon) \log \log n$ *adaptive queries.*

3.3 Lower Bound for Sorted List with Negative Elements

We derive a theorem that shows there is not any factor approximation sublinear time algorithm for the sum of a list of elements that contains both positive and negative elements.

Theorem 4. *Let* ϵ *be an arbitrary positive constant. There is no algorithm that makes at most* $n - 1$ *queries to give* $(1 + \epsilon)$*-approximation for the sum of a list of* n *sorted elements that contains at least one negative element.*

Proof. Consider a list of element $-m(m + 1), 2, \cdots, 2m$. This list contains $n = m + 1$ elements. If there is an algorithm that gives $(1 + \epsilon)$-approximation, then there is an element, say $2k$, that is not queried by the algorithm.

We construct another list that is identical to the last list except $2k$ being replaced by $2k + 1$.

The sum of the first list is zero, but the sum of the second list is 1. The algorithm gives the same result as the element $2k$ in the first list and the element $2k+1$ in the second list are not queried (all the other queries are the of the same answers). This brings a contradiction.

Similarly, in the case that $-m(m + 1)$ is not queried, we can bring a contradiction after replacing it with $-m(m + 1) + 1$.

4 Conclusions and Open Problems

We studied the approximate sum in a sorted list with nonnegative elements. For a fixed ϵ, there is a $\log \log n$ factor gap between the upper bound of our algorithm, and our lower bound. An interesting problem of further research is to close this gap. Another interesting problem is the computational complexity of approximate sum in the randomized computational model, which is not discussed in this paper.

Acknowledgments. The author would like to thank Cynthia Fu for her proofreading and comments for an earlier version of this paper, and anonymous referees for providing comments to improve the presentation of this paper.

References

1. Anderson, I.J.: A distillation algorithm for floating-point summation. SIAM J. Sci. Comput. 20, 1797–1806 (1999)
2. Bresenham, J.E.: Algorithm for computer control of a digital plotter. IBM Systems Journal 4(1), 25 (1965)
3. Canetti, R., Even, G., Goldreich, O.: Lower bounds for sampling algorithms for estimating the average. Information Processing Letters 53, 17–25 (1995)
4. Demmel, J., Hida, Y.: Accurate and efficient floating point summation. SIAM J. Sci. Comput. 25, 1214–1248 (2003)
5. Espelid, T.O.: On floating-point summation. SIAM Rev. 37, 603–607 (1995)
6. Gregory, J.: A comparison of floating point summation methods. Commun. ACM 15, 838 (1972)
7. Har-Peled, S.: Coresets for discrete integration and clustering. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 33–44. Springer, Heidelberg (2006)
8. Higham, N.J.: The accuracy of floating point summation. SIAM J. Sci. Comput. 14, 783–799 (1993)
9. Hoefding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association 58, 13–30 (1963)
10. Kahan, W.: Further remarks on reducing truncation errors. Communications of the ACM 8(1), 40 (1965)
11. Knuth, D.E.: The art of computer programming, 3rd edn. Seminumerical Algorithms, vol. 2. Addison-Wesley, Reading (1998)
12. Linz, P.: Accurate floating-point summation. Commun. ACM 13, 361–362 (1970)
13. Malcolm, M.A.: On accurate floating-point summation. Commun. ACM 14, 731–736 (1971)
14. Motwani, R., Panigrahy, R., Xu, Y.: Estimating sum by weighted sampling. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 53–64. Springer, Heidelberg (2007)
15. Priest, D.M.: On Properties of Floating Point Arithmetics: Numerical Stability and the Cost of Accurate Computations, Ph.D. thesis. PhD thesis, Mathematics Department, University of California, Berkeley, CA (1992)
16. Zhu, Y.K., Yong, J.H., Zheng, G.Q.: A new distillation algorithm for floating-point summation. SIAM Journal on Scientific Computing 26, 2066–2078 (2005)

Large Hypertree Width for Sparse Random Hypergraphs

Chaoyi Wang[1], Tian Liu[1,⋆], and Ke Xu[2,⋆]

[1] Key Laboratory of High Confidence Software Technologies, Ministry of Education, Institute of Software, School of Electronic Engineering and Computer Science, Peking University, Beijing 100871, China
lt@pku.edu.cn

[2] National Lab. of Software Development Environment, Beihang University, Beijing 100191, China
kexu@nlsde.buaa.edu.cn

Abstract. Hypertree width is a similar notion to treewidth, also with many equivalent characterizations and many applications. If the hypertree widths of constraint graphs of instances of Constraint Satisfaction Problems (CSPs) are bounded by a constant, the relevant constraint satisfaction problems are tractable. In this paper, we show that with high probability, hypertree width is large on sparse random k-uniform hypergraphs. Our results provide further theoretical evidence on hardness of some random constraint satisfaction problems, called Model RB and Model RD, around the satisfiability phase transition points.

Keywords: Constraint Satisfaction, Random Hypergraph, Hypertree Width, Model RB and Model RD, Phase Transition.

1 Introduction

Constraint Satisfaction Problems (CSPs) can represent many important NP-hard problems in Artificial Intelligence research. In the last two decades, there are two lines of research on CSPs, both are fruitful. The first one is about structural decompositions; the second one is about random CSPs.

A CSP instance is called *acyclic* if its constraint hypergraph is acyclic [24]. Checking the satisfiability of acyclic CSPs is tractable [24]. Many CSP instances arising in practice are not acyclic, but are in some sense close to acyclic ones. Structural decomposition methods are proposed to measure the difference between acyclic hypergraphs and the others. That is, the smaller the parameter of the structural decomposition is, the more similar the hypergraph is to an acyclic one. The most prominent decomposition methods include: tree clustering [20, 10, 6], hinge decomposition [17, 18, 22], cycle cutset and cycle hypercutset [5, 15], hinge-tree clustering [15], query decomposition [3], and (generalized) hypertree decomposition [16, 13, 1]. For structural decompositions of CSPs, the

⋆ Corresponding authors. Partially supported by National 973 Program of China (Grant No. 2010CB328103).

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 294–302, 2013.

main results say that CSPs with constraint hypergraphs of bounded structural width are tractable. In particular, all known structural decomposition based solvers run in time $||I||^{O(w)}$, where $||I||$ is the input size and w is a structural width, such as the size of the minimum loop-cutset, hinge width, treewidth, query width, hypertree width, etc. A plethora of structural decomposition methods have been developed and compared with each other, see e.g. [14].

On the other hand, the start point of random CSPs is to identify hard instances to serve as benchmarks, such as random instances around the satisfiability threshold of random CSPs [2, 4, 7–9, 23, 25, 27–29]. The main results say that there is an easy-hard-easy transition around the satisfiability threshold of random CSPs, and the hardest instances are around the satisfiability thresholds.

However, a rigorous link between phase transition and hardness of random instances is hard to establish. At the moment, we still do not know whether there exist polynomial solvers at the satisfiability thresholds. But we can show some theoretical evidence on hardness of random instances for some specific solvers that cannot solve the CSPs efficiently.

The most popular structural width is treewidth [20]. Kloks showed a large treewidth on classical Erdős-Rényi random graphs $G(n, p)$ when $c = np > 2.36$. After that, a lot of results related to large treewidth and smaller c have been proposed. Now it's clear that when $c > 1$ is a constant, treewidth is large with high probability [21]. It is also known that treewidth is equal to $n - o(n)$ if and only if c is unbounded [26]. To investigate the tractability of random CSPs and Bayesian networks, Gao initiated the study of treewidth in various kinds of random (hyper)graphs [10–12].

To show the hardness of random CSPs, solvers based on different structural decompositions have been considered. It is known that hinge width on sparse k-uniform random hypergraphs is large, which implies that solvers based on hinge decomposition cannot solve the relevant CSPs efficiently [22], and [19] shows that the size of minimum loop-cutset is large, which also gives some theoretical evidence on the hardness of random CSPs.

(Generalized) hypertree decomposition is almost the most general of such decomposition methods leading to large tractable classes of CSPs [14]. It is also one of the most powerful decomposition methods for CSPs in practice. In addition, Hypertree decomposition plays a similar role for hypergraphs as tree decomposition for graphs. Up to a constant factor, hypertree width is the same as a number of other hypergraph invariants such as bramble number, branch width, linkedness, and the minimum number of cops required to win Seymour and Thomas's robber and cops game [1]. Compared with other parameters, hypertree width has many advantages [13, 14], such as there is a polynomial algorithm for checking whether the hypertree width of the given hypergraph is at most k, where k is a constant [13], and hypertree width is smaller than treewidth and hinge width on arbitrary hypergraphs, so it leads to a larger tractable class of CSP instances [14].

As far as we know, hypertree width on random hypergraphs is unknown. In this paper we show that *hypertree width* is large with high probability for sparse random k-uniform hypergraphs, where k is a constant. By the comparison results of different structural decomposition methods [14], we can conclude that the large hypertree width will mean that almost all structural parameters are large, that is why the random CSPs are so difficult to solve by using structural decomposition solvers.

This paper is structured as follows. After introducing necessary definitions and notations (Section 2), lower bounds on hypertree width for random hypergraphs are shown (Section 3). Finally are some concluding remarks (Section 4).

2 Preliminaries

In this section, we give necessary definitions and notations.

Definition 1. (Constraint Satisfaction Problems) *A constraint satisfaction problem (CSP) instance is a triple* $(\mathcal{V}, \mathcal{D}, \mathcal{C})$*, where*

- $\mathcal{V} = \{v_1, \ldots, v_n\}$ *is a set of* n *variables;*
- $\mathcal{D}$ *is a finite set of values called* domain*, whose size* $|\mathcal{D}|$ *can be either fixed or increasing with* n*;*
- $\mathcal{C}$ *is a set of constraints, and each constraint* C_i *is a pair* (S_i, R_i)*, where*
  - S_i *is a* k*-tuple of variables, called* constraint scope*;*
  - R_i *is a subset of* $\mathcal{D}^k$*, called* constraint relation*.*

A constraint $C_i = (S_i, R_i)$ *is* satisfied *if the* k*-tuple of values assigned to variables in* S_i *is contained in* R_i*. A* solution *to a CSP instance is an assignment of values to all the variables that satisfies all constraints.*

Definition 2. (Model RB and Model RD [27]) *Model RB is a random CSP model defined as:*

- *given* n *variables each with domain* $\{1, 2, ..., d\}$*, where* $d = n^\alpha$ *and* $\alpha > 0$ *is constant;*
- *select with repetition* $m = rn \ln n$ *random constraints, for each constraint select without repetition* k *of* n *variables, where* $k \geq 2$ *is an integer constant;*
- *select uniformly at random without repetition* $(1-p)d^k$ *compatible assignments for each constraint, where* $0 < p < 1$ *is constant.*

If in the last step above, each assignment (out of the total d^k *assignments) for the* k *variables is selected with probability* $1-p$ *as compatible independently, then it is called Model RD.*

In the following, let $\mathcal{H} = (V, \mathcal{E})$ be a *hypergraph*, consisting of a set V of vertices, and a set $\mathcal{E}$ of subsets of V. We call the element in $\mathcal{E}$ as *hyperedges*. The *constraint hypergraph* of a CSP instance is a hypergraph with variables as vertices and constraint scopes as hyperedges. The constraint hypergraphs of random CSPs are random hypergraphs.

Definition 3. (Random Hypergraphs $G(n,p,k)$ [22]) *We use $G(n,p,k)$ to denote the probability space of k-uniform random hypergraphs, in which on n vertices, each k-element subset of vertices is selected with probability p independently at random as a hyperedge, and k is a fixed positive integer. When the total number of hyperedges is $O(n)$ or $O(n \ln n)$, these hypergraphs are called* sparse.

Definition 4. (Treewidth [20, 10, 1]) *A* tree decomposition *of a hypergraph $\mathcal{H}$ is a pair (T,χ), where $T=(N,F)$ is a tree and χ is a labeling function which associates to each vertex $p \in N$ a set $\chi(p) \subseteq V$, we call these sets as* bags. *The following conditions should also be satisfied. For each $e \in \mathcal{E}$, there is a bag $\chi(p)$ such that $e \subseteq \chi(p)$, and for each $v \in V$, the set $\{p \in N \mid v \in \chi(p)\}$ induces a connected subtree in T. The* width *of the decomposition (T,χ) is* $\max_{p\in N}(|\chi(p)|-1)$, *and the* treewidth *of $\mathcal{H}$, denoted by $tw(\mathcal{H})$, is the minimum of the widths of all tree decompositions of $\mathcal{H}$, that is*

$$tw(\mathcal{H}) = \min_{(T,\chi)} \max_{p\in N}(|\chi(p)|-1).$$

The *primal graph* [14] of a hypergraph $\mathcal{H}=(V,\mathcal{E})$, denoted by $G(\mathcal{H})=(V,E)$, is the graph whose edge set is $E=\{(v_1,v_2) \mid v_1,v_2 \in V$, and there exists $h \in \mathcal{E}$, such that $v_1,v_2 \in h\}$, i.e. each hyperedge forms a clique. Since a graph is also a hypergraph, we also denote the *treewidth* of a graph G as $tw(G)$. Since every clique must appear in at least one bag in a tree decomposition, the *treewidth* $tw(\mathcal{H})$ of a hypergraph $\mathcal{H}$ is equal to the *treewidth* $tw(G(\mathcal{H}))$ of its primal graph $G(\mathcal{H})$, i.e. $tw(\mathcal{H})=tw(G(\mathcal{H}))$.

A *hypertree* for a hypergraph $\mathcal{H}=(V,\mathcal{E})$ [16] is a triple (T,χ,λ), where $T=(N,F)$ is a rooted tree directed from root to the leaves, and χ and λ are labeling functions which associate to each vertex $p \in N$ two sets $\chi(p) \subseteq V$ and $\lambda(p) \subseteq \mathcal{E}$. If $T'=(N',F')$ is a subtree of T, we define $\chi(T')=\bigcup_{v\in N'}\chi(v)$. In addition, for any $p \in N$, T_p denotes the subtree of T rooted at p, that is, the induced subtree of T whose vertex set is the set of all vertices reachable from p.

Definition 5. (Hypertree Width [13, 24, 1]) *A* generalized hypertree decomposition *of a hypergraph $\mathcal{H}$ is a hypertree $HT=(T,\chi,\lambda)$, where (T,χ) is a tree decomposition of $G(\mathcal{H})$ and for every $p \in V(T)$, we have $\chi(p) \subseteq \bigcup\lambda(p)$. The width of the hypertree HT is* $\max_{p\in V(T)}|\lambda(p)|$. *The* generalized hypertree width *of $\mathcal{H}$, denoted by $ghw(\mathcal{H})$, is the minimum of the widths of all the generalized hypertree decompositions of $\mathcal{H}$, that is*

$$ghw(\mathcal{H}) = \min_{(T,\chi,\lambda)} \max_{p\in V(T)} |\lambda(p)|.$$

A hypertree decomposition *of a hypergraph $\mathcal{H}$ [16] is a generalized hypertree decomposition $HT=(T,\chi,\lambda)$ which satisfies the following special condition:*

$$(\bigcup\lambda(p)) \cap \chi(T_p) \subseteq \chi(p) \text{ for all } p \in V(T).$$

The hypertree width *of $\mathcal{H}$, denoted by $hw(\mathcal{H})$, is the minimum of the widths of all hypertree decompositions of $\mathcal{H}$.*

Actually, there is a relation between $tw(\mathcal{H})$, $ghw(\mathcal{H})$, and $hw(\mathcal{H})$, see [1],

$$ghw(\mathcal{H}) \leq hw(\mathcal{H}) \leq tw(\mathcal{H}) + 1.$$

For the k-uniform hypergraph $G(n, p, k)$, since one hyperedge can cover at most k vertices, we have

$$\frac{tw(G(n,p,k)) + 1}{k} \leq ghw(G(n,p,k)).$$

In the next section, we will give a lower bound on $hw(G(n, p, k))$ by giving a lower bound on $tw(G(n, p, k))$.

Denote by $\Pr(\mathcal{E})$ the probability of an event $\mathcal{E}$, $\mathbb{E}X$ or $\mathbb{E}(X)$ the expectation of a random variable X, $\mathbf{B}(n, p)$ the binomial distribution. An event $\mathcal{E}$ occurs *with high probability* (**whp**), if $\lim_{n\to\infty} \Pr(\mathcal{E}) = 1$. Both $f \ll g$ and $f = o(g)$ mean $\lim_{n\to\infty} \frac{f}{g} = 0$. A useful inequality is $1 - x < e^{-x}$ for $x > 0$.

3 Lower Bound on Hypertree Width for Random Hypergraphs

We now calculate treewidth on $G(n, p, k)$.

Lemma 1. *[20] Let $G = (V, E)$ be a graph with at least $k + 1$ vertices and $tw(G) \leq k$. Then there exists a set S of $k+1$ vertices such that every component of $G[V\backslash S]$ has at most $(n - k)/2$ vertices.*

Lemma 2. *Let $G = (V, E)$ be a graph with at least $k+1$ vertices and $tw(G) \leq k$. Then there exists a set S of $k + 1$ vertices and three disjoint sets A, B, C, such that $A \cup B \cup C = V\backslash S$, and there is no edge between A,B,C, and each of the three sets has at most $(n - k)/2$ vertices.*

Proof. Assume that $G[V\backslash S]$ has t components $X_1, \cdots, X_t$. By Lemma 1,

$$|X_i| \leq (n - k)/2 \text{ for } i = 1, \cdots, t.$$

If $t = 3$, X_1, X_2, X_3 satisfy the condition, so $A = X_1$, $B = X_2$, $C = X_3$. If $n \geq 4$, without loss of generality, let X_{t-1} and X_t be the two smallest components. Let $X'_{t-1} = X_t \cup X_{t-1}$. Since $|X_{t-1}| + |X_t| \leq n - |S| - |X_1| - |X_2|$, $|X_{t-1}| \leq |X_1|$, $|X_t| \leq |X_2|$, we get $2(|X_{t-1}| + |X_t|) \leq n - |S|$, i.e. $|X'_{t-1}| \leq (n - k)/2$. So we get $t - 1$ disjoint sets, $X_1, \ldots, X_{t-2}, X'_{t-1}$, each of them has at most $(n - k)/2$ vertices and there is no edge between them. Repeat the above steps $t - 3$ times, we can get the three sets X_1, X_2 and X'_3, and $A = X_1$, $B = X_2$, $C = X'_3$. □

Lemma 3. *Let $G \in G(n, p = \frac{c}{n^{k-1}}, k)$, $G = (V, E)$, $|V| = n$, $0 < t < 1$. Let*

$$f(t) = k! \frac{(1-t)(1 - \ln(1-t) + \ln 3) + \epsilon}{1 + t^k - (1+t)^k/2^{k-1}},$$

where $\epsilon > 0$ is a constant. If $c > f(t)$, then $tw(G) \geq tn$ with high probability.

Proof. Let $P = (S, A, B, C)$ is a partition of V, where $|S| = tn$, and A, B, C has at most $\frac{1}{2}(n - tn - 1)$ vertices. Let $\mathcal{P}$ denote the set of all the partitions satisfying the above conditions. We call a partition P as a $S-separator$ if there is no hyperedge between A, B, C. For any partition $P \in \mathcal{P}$, define a random variable as follows:

$$I_p = \begin{cases} 1, P \text{ is a } S\text{-separator} \\ 0, \text{otherwise} \end{cases}$$

Since every edge which is not between A, B, C is contained in $A \cup S$,$B \cup S$ or $C \cup S$, according to the inclusion-exclusion principle, the number of such edges is

$$\binom{|A|+|S|}{k} + \binom{|B|+|S|}{k} + \binom{|C|+|S|}{k} - 2\binom{|S|}{k}$$

So, the number m' of potential edges between A, B, C is

$$\binom{n}{k} - \left[\binom{|A|+|S|}{k} + \binom{|B|+|S|}{k} + \binom{|C|+|S|}{k} - 2\binom{|S|}{k}\right].$$

Now, we give a lower bound on m'. Define a function

$$h(x, y) = \binom{x}{k} + \binom{y}{k},$$

and suppose that $x + y = u$ is a constant. Then, we can find that $h(x, y)$ will increase if we increase the gap between x and y. So, let $|S| = s$,

$$\begin{aligned} &\binom{|A|+|S|}{k} + \binom{|B|+|S|}{k} + \binom{|C|+|S|}{k} - 2\binom{|S|}{k} \\ &< \binom{\frac{|V|-|S|}{2}+|S|}{k} + \binom{\frac{|V|-|S|}{2}+|S|}{k} + \binom{|S|}{k} - 2\binom{|S|}{k} \\ &= 2\binom{\frac{n+s}{2}}{k} - \binom{s}{k}. \end{aligned}$$

So the number of potential edges between A, B, C is at least

$$\binom{n}{k} - 2\binom{\frac{n+s}{2}}{k} + \binom{s}{k}.$$

When n is sufficiently large, we have

$$\binom{n}{k} \geq \frac{n^k - \binom{k}{2}n^{k-1}}{k!}, \quad \binom{s}{k} \geq \frac{s^k - \binom{k}{2}s^{k-1}}{k!}, \quad \binom{\frac{n+s}{2}}{k} \leq \frac{(\frac{n+s}{2})^k}{k!}.$$

Putting them all together and substituting $s = tn$, we have

$$\begin{aligned} &\binom{n}{k} - 2\binom{\frac{n+s}{2}}{k} + \binom{s}{k} \\ &\geq \left(n^k - \binom{k}{2}n^{k-1} - 2\left(\frac{n+s}{2}\right)^k + s^k - \binom{k}{2}s^{k-1}\right)/k! \\ &= \left(\left(1+t^k\right)n^k - \binom{k}{2}\left(1+t^{k-1}\right)n^{k-1} - \frac{1}{2^{k-1}}(1+t)^k n^k\right)/k!. \end{aligned}$$

So, $\mathbb{E}(I_p) = (1-p)^{m'} \leq e^{-pm'} < e^{-\frac{c}{k!}\left(n(1+t^k-\frac{1}{2^{k-1}}(1+t)^k)-\binom{k}{2}(1+t^{k-1})\right)}$.

Let $I = \sum_{P\in\mathcal{P}} I_P$. Then $I = 0$ means every partition in $\mathcal{P}$ is not a $S-$*separator*, that is, $tw(G) \geq tn$. By the linearity of expectation and the union bound, the expectation of I is

$$\begin{aligned}\mathbb{E}(I) &= \textstyle\sum_{P\in\mathcal{P}} \mathbb{E}(I_P) < \binom{n}{tn} 3^{n-tn}\mathbb{E}(I_P) < (\tfrac{en}{(1-t)n})^{(1-t)n} 3^{n-tn}\mathbb{E}(I_P) \\ &= e^{-n\left(\frac{c}{k!}(1+t^k-\frac{1}{2^{k-1}}(1+t)^k)-(1+\ln 3-\ln(1-t))(1-t)\right)+\frac{c}{k!}\binom{k}{2}(1+t^{k-1})}.\end{aligned}$$

If $c > f(t)$,

$$\Pr(I > 0) \leq \mathbb{E}(I) < e^{-\epsilon n+\frac{c}{k!}\binom{k}{2}(1+t^{k-1})} = o(1).$$

So, $tw(G) \geq tn$ whp. □

Corollary 1. *For the special case $G(n,p,2)$, which is the classical Erdös – Rényi random graph model $G(n,p)$, if $c > \frac{4(1-\ln(1-t)+\ln 3)+\epsilon}{1-t}$, then $tw(G) \geq tn$ whp.*

Proof. We only need to substitute $k = 2$ to the Lemma 3. Note that this is an asymptotically better result than that in [26]. □

From Lemma 3, we can get two theorems as follows.

Theorem 4. *Let $G \in G(n, p = \frac{c}{n^{k-1}}, k)$, $G = (V,E)$, $|V| = n$,$0 < t < 1$, let*

$$f(t) = k!\frac{(1-t)(1-\ln(1-t)+\ln 3)+\epsilon}{1+t^k-(1+t)^k/2^{k-1}}$$

where $\epsilon > 0$ is a constant, if $c > f(t)$, then $hw(G) \geq ghw(G) \geq \frac{tn+1}{k}$ whp.

Proof. By Lemma 3, and the fact that generalized hypertree width is based on treewidth and only use hyperedges to cover the bags of the tree decomposition. Since each hyperedge covers at most k vertices, so $ghw(G) \geq \frac{tn+1}{k}$. Note that hypertree decomposition is a special generalized hypertree decomposition, we have $hw(G) \geq ghw(G)$. □

Note actually the edges couldn't cover the vertices so efficiently, but in the complete graph, which has the most edges in the k-uniform hypergraph, it indeed can achieve this lower bound. Since the hypertree width is not a monotone variable, so it's an open problem to find the gap between the lower bound and hypertree width.

Theorem 5. *Let $G \in G(n, p = \frac{r\ln n}{n^{k-1}}, k)$, $G = (V,E)$, $|V| = n$, r is a constant. Then $hw(G) = \frac{n-o(n)}{k}$ whp.*

Proof. For any constant $t < 1$, $r\ln n > f(t)$, which implies the above result. □

By Theorem 5, the random instances of Model RB and Model RD around the satisfiability thresholds $r_{cf} = -\frac{\alpha}{\ln(1-p)}$ [27] has a large *hypertree width*. Therefore, a large *hypertree width* around the satisfiability threshold can provide some theoretical evidence for hardness of Model RB and Model RD besides the known theoretical hardness evidences based on their exponential resolution complexity [28], large loop cueset [19], large hinge width [22], and large treewidth [26].

4 Conclusion and Open Problems

In the above section, we show a linear hypertree width in sparse random k-uniform hypergraphs. When the number of hyperedges is super linear, we can get a linear hypertree width which almost depends only on k, which implies the hypertree decomposition based solvers can't solve the random CSPs, such as Model RB and Model RD around their thresholds efficiently. When the number of hyperedges is linear to the vertices, the linear hypertree width depends on k and the number of edges. So it is an open problem to find the exact point when the hypertree width becomes $O(n)$. Since the hypertree width of the complete k-uniform hypergraph which has the most edges is n/k, whether there is some graph with a larger hypertree width is also unknown.

Acknowledgments. We thank Professor Kaile Su for his encouragement and support to this work. We also thank the unknown reviewers of FAW-AAIM 2013 for helpful comments.

References

1. Adler, I., Gottlob, G., Grohe, M.: Hypertree width and related hypergraph invariants. European Journal of Combinatorics 28, 2167–2181 (2007)
2. Cheeseman, P., Kanefsky, R., Taylor, W.: Where the really hard problems are. IJCAI 1991, 163–169 (1991)
3. Chekuri, C., Rajaraman, A.: Conjunctive query containment revisited. Theoretical Computer Science 239(2), 211–229 (2000)
4. Cook, S.A., Mitchell, D.G.: Finding hard instances of the satisfiability problem: a survey. DIMACS Series, vol. 35, pp. 1–17 (1997)
5. Dechter, R.: Constraint networks. In: Encyclopedia of Artificial Intelligence, 2nd edn., vol. 1, pp. 276–285. Wiley and Sons, Chichester (1992)
6. Dechter, R., Pearl, J.: Tree clustering for constraint networks. Artificial Intelligence 38(3), 353–366 (1989)
7. Fan, Y., Shen, J.: On the phase transitions of random k-constraint satisfaction problems. Artificial Intelligence 175, 914–927 (2011)
8. Fan, Y., Shen, J., Xu, K.: A general model and thresholds for random constraint satisfaction problems. Artificial Intelligence 193, 1–17 (2012)
9. Gao, Y., Culberson, J.: Consistency and random constraint satisfaction problems. J. Artif. Intell. Res. (JAIR) 28, 517–557 (2007)
10. Gao, Y.: Phase transition of tractability in constraint satisfaction and Bayesian network inference. Proc. UAI, 265–271 (2003)
11. Gao, Y.: On the threshold of having a linear treewidth in random graphs. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 226–234. Springer, Heidelberg (2006)
12. Gao, Y.: Treewidth of Erdös-Rényi random graphs, random intersection graphs, and scale-free random graphs. CoRR abs/0907.5481 (2009)
13. Gottlob, G., Leone, N., Scarcello, F.: Hypertree Decompositions: A Survey. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, p. 37. Springer, Heidelberg (2001)

14. Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. Artificial Intelligence 124, 243–282 (2000)
15. Gottlob, G., Leone, N., Scarcello, F.: The complexity of acyclic conjunctive queries. Journal of ACM 48(3), 431–498
16. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decomposition and tractable queries. Journal of Computer and System Sciences 64(3), 579–627 (2002)
17. Gyssens, M., Jeavons, P.G., Cohen, D.A.: Decomposition constraint satisfaction problems using database techniques. Artificial Intelligence 66(1), 57–89 (1994)
18. Gyssens, M., Paredaens, J.: A decomposition methodology for cyclic databases. In: Gallaire, H., Nicolas, J.-M., Minker, J. (eds.) Advances in Data Base Theory, vol. 2, pp. 85–122. Plemum Press, New York (1984)
19. Jiang, W., Liu, T., Ren, T., Xu, K.: Two hardness results on feedback vertex sets. In: Atallah, M., Li, X.-Y., Zhu, B. (eds.) FAW-AAIM 2011. LNCS, vol. 6681, pp. 233–243. Springer, Heidelberg (2011)
20. Kloks, T.: Treewidth: Computations and Approximations, page 18, 55. Springer (1994)
21. Lee, C., Lee, J., Oum, S.: Rank-width of random graphs. CoRR abs/1001.0461 (2010)
22. Liu, T., Lin, X., Wang, C., Su, K., Xu, K.: Large Hinge Width on Sparse Random Hypergraphs. IJCAI 2011, 611–616 (2011)
23. Mitchell, D.G., Selman, B., Levesque, H.J.: Hard and easy distributions of sat problems. In: Proc. AAAI, pp. 459–465 (1992)
24. Dechter, R.: Tractable Structures for Constraint Satisfaction Problems. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, pp. 209–244. Elsevier, Amsterdam (2006)
25. Selman, B., Mitchell, D.G., Levesque, H.J.: Generating hard satisfiability problems. Artif. Intell. 81, 17–29 (1996)
26. Wang, C., Liu, T., Cui, P., Xu, K.: A Note on Treewidth in Random Graphs. In: Wang, W., Zhu, X., Du, D.-Z. (eds.) COCOA 2011. LNCS, vol. 6831, pp. 491–499. Springer, Heidelberg (2011)
27. Xu, K., Li, W.: Exact phase transitions in random constraint satisfaction problems. J. Artif. Intell. Res. 12, 93–103 (2000)
28. Xu, K., Li, W.: Many hard examples in exact phase transitions. Theor. Comput. Sci. 355, 291–302 (2006)
29. Xu, K., Boussemart, F., Hemery, F., Lecoutre, C.: Random Constraint Satisfaction: Easy Generation of Hard (Satisfiable) Instances. Artif. Intell. 171, 514–534 (2007)

On Perfect Absorbants in De Bruijn Digraphs

Yue-Li Wang[1,⋆], Kuo-Hua Wu[1], and Ton Kloks[2,⋆⋆]

[1] Department of Information Management
National Taiwan University of Science and Technology
No. 43, Sec. 4, Keelung Rd., Taipei, 106, Taiwan
ylwang@cs.ntust.edu.tw
[2] Department of Computer Science
National Tsing Hua University
No. 101, Sec. 2, Kuang Fu Rd., Hsinchu, Taiwan

Abstract. An absorbant of a digraph D is a set $S \subseteq V(D)$ such that, for every $v \in V(D) \setminus S$, there exists an arc (v,u) with $u \in S$. We denote the cardinality of a minimum absorbant by $\gamma_a(D)$. An absorbant S is called a perfect absorbant if no vertex of S has an out-neighbor in S and no two vertices in S have a common in-neighbor.

In this paper, we are concerned with the perfect absorbant problem in generalized De Bruijn digraphs. We prove that some classes of generalized De Bruijn graphs have perfect absorbants. We also answer some questions asked by Shan et al. in [10], i.e., we affirm that $\gamma_a(G_B(8k-4,4k-3)) = 3$ and $\gamma_a(G_B(6k,2k-1)) = 4$ for $k \geqslant 2$.

Keywords: Domination, Absorbant, Upper bound, Generalized De Bruijn digraphs.

1 Introduction

Let $D = (V,A)$ denote a digraph in which V and A are the sets of vertices and arcs. The *in-neighborhood* and *out-neighborhood* of a vertex $v \in V$, denoted by $I(v)$ and $O(v)$ respectively, are the sets

$$I(v) = \{\, u \mid (u,v) \in A \,\} \quad \text{and} \quad O(v) = \{\, u \mid (v,u) \in A \,\}.$$

We allow *self-loops*, i.e. arcs of the form (u,u) for some $u \in V$. The *closed in-neighborhood* and *closed out-neighborhood* of v are defined as

$$I[v] = I(v) \cup \{v\} \quad \text{and} \quad O[v] = O(v) \cup \{v\}.$$

A *dominating set* in a digraph $D = (V,A)$ is a set $S \subseteq V$ such that, for every $v \in V \setminus S$, there exists an arc $(u,v) \in A$ with $u \in S$. The *domination number* of D, which we denote by $\gamma(D)$, is defined as the minimal cardinality of a dominating set of D.

⋆ National Science Council of Taiwan Support Grant NSC100–2221–E–011–067-MY3.
⋆⋆ National Science Council of Taiwan Support Grant NSC 99–2218–E–007–016.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 303–314, 2013.

An *absorbant* of a digraph $D = (V, A)$ is a set $S \subseteq V$ such that, for every $v \in V \setminus S$, there exists an arc $(v, u) \in A$ with $u \in S$. The *absorbant number* of D, denoted by $\gamma_a(D)$, is defined as the minimum cardinality of an absorbant of D. A dominating set (respectively, absorbant) S in D is *perfect* if $O(u) \cap O(v) = \varnothing$ (respectively, if $I(u) \cap I(v) = \varnothing$) for any two distinct vertices $u, v \in S$ and there is no arc in the subdigraph induced by S.

Generalized De Bruijn digraphs were introduced by Imase and Itoh [5, 6] and, independently, also by Reddy, Pradhan, and Kuhl [9]. The generalized De Bruijn digraph $G_B(n, d)$ has vertex set $V(G_B(n, d)) = \{0, 1, 2, \ldots, n-1\}$ and arc set

$$A(G_B(n, d)) = \{ (x, y) \mid y \equiv dx + i \pmod{n} \quad 0 \leqslant i < d \}. \tag{1}$$

Note that if $n = d^m$, then $G_B(n, d)$ is the well-known De Bruijn digraph $B(d, m)$.

De Bruijn digraphs have good logical network topologies, comparable to hypercubes, that make them suitable for interconnection networks [1, 3–5, 9]. The connectivity and diameter of $G_B(n, d)$ were investigated in [2, 5, 7, 9]. In [2], Du and Hwang prove that the number of self-loops in $G_B(n, d)$ is equal to $g \cdot \lfloor \frac{d}{g} \rfloor$, where $g = \gcd(n, d-1)$. They also show that $G_B(n, d)$ is Hamiltonian if $\gcd(n, d) > 1$. In [8], Kikuchi and Shibata show that

$$\left\lceil \frac{n}{d+1} \right\rceil \leqslant \gamma(G_B(n, d)) \leqslant \left\lceil \frac{n}{d} \right\rceil,$$

and Shan, Cheng, and Kang show that the same bounds hold for $\gamma_a(G_B(n, d))$ in [10].

We are interested in perfect absorbants and perfect dominating sets of generalized De Bruijn graphs. This paper is organized as follows. In Section 2 we review some preliminaries of perfect absorbants in generalized De Bruijn digraphs. In Section 3 we classify generalized De Bruijn digraphs into four classes and show that there exist perfect absorbants for the digraphs in two of those classes. We conclude in Section 5.

2 Preliminaries

When the digraph D is clear from the context we write V and A instead of $V(D)$ and $A(D)$. A generalized De Bruijn digraph $G_B(n, d)$ can have a perfect absorbant only if $d > 1$ and $n = c(d+1)$ for some integer c. Henceforth, we assume that the parameters n and d satisfy this condition.

The following two theorems give lower and upper bounds for dominating sets and absorbants in $G_B(n, d)$, respectively.

Theorem 1 ([8]). $\lceil \frac{n}{d+1} \rceil \leqslant \gamma(G_B(n, d)) \leqslant \lceil \frac{n}{d} \rceil$.

Theorem 2 ([10]). $\lceil \frac{n}{d+1} \rceil \leqslant \gamma_a(G_B(n, d)) \leqslant \lceil \frac{n}{d} \rceil$.

Via a standard counting argument it follows from the definitions that, if there exists a perfect dominating set or a perfect absorbant in $G_B(n,d)$, then

$$\boxed{\gamma(G_B(n,d)) = \gamma_a(G_B(n,d)) = c = \frac{n}{d+1}}$$

By definition, an absorbant S is perfect if and only if the following three properties hold:

(i) For any two distinct vertices $u, v \in S$, $I(u) \cap I(v) = \varnothing$.
(ii) For any two distinct vertices $u, v \in S$, $u \notin I(v)$ and $v \notin I(u)$.
(iii) For each vertex $v \in S$, $v \notin I(v)$.

Note that Property (i) ensures that the in-neighborhoods of any two distinct vertices in S are disjoint while Properties (ii) and (iii) ensure that there is no arc or self-loop in the subdigraph induced by S.

For a vertex $u \in O(v)$ with $u \equiv dv + j \pmod{n}$ and $0 \leqslant j < d$, we call u the (unique) j^{th} *out-neighbor* of v and we denote it by $O_j(v)$.

The following lemma can be used to determine if $x \in I(x)$ in $G_B(n,d)$.

Lemma 1. *For a vertex* x *in* $G_B(n,d)$,

$$x \in I(x) \text{ if and only if } 0 \leqslant -(d-1)x \pmod{n} \leqslant d-1.$$

Proof. By Equation (1),

$$x \in I(x) \text{ if and only if } x = dx + i + pn \text{ for some } 0 \leqslant i < d \text{ and integer } p.$$

This implies that $x \in I(x)$ if and only if $-(d-1)x \equiv i \pmod{n}$.

If we use mod as an operator we obtain, since $i \in \{0, 1, ..., d-1\}$,

$$x \in I(x) \text{ if and only if } 0 \leqslant -(d-1)x \pmod{n} \leqslant d-1.$$

This completes the proof. □

Let $L_0(n,d) = \{x | x = O_0(x), x \in V\}$ and $L_{d-1}(n,d) = \{x | x = O_{d-1}(x), x \in V\}$. When there is no possible ambiguity we use L_0 and L_{d-1} instead of $L_0(n,d)$ and $L_{d-1}(n,d)$.

Lemma 2. *If* $n = c(d+1)$, *then* $|L_0(n,d)| = |L_{d-1}(n,d)| = \gcd(2c, d-1)$.

Proof. We only prove that $|L_0(n,d)| = \gcd(2c, d-1)$. The other case is similar. By Lemma 1, $x = O_0(x)$ if and only if $0 \equiv (d-1)x \pmod{n}$. Thus $|L_0|$ is equal to the number of x for $0 \leqslant x \leqslant n-1$ such that $(d-1)x = yn$ for integer $y \geqslant 0$. The above equation can be simplified as $x = \frac{yn}{d-1} = \frac{yc(d+1)}{d-1} = cy + \frac{2cy}{d-1}$. Let $g = \gcd(2c, d-1)$, $2c = z_1 g$ and $d-1 = z_2 g$. Thus $x = cy + \frac{2cy}{d-1} = cy + \frac{gz_1 y}{gz_2} = cy + \frac{z_1 y}{z_2}$. Since the term $\frac{z_1 y}{z_2}$ must be a nonnegative integer, y is a multiple of z_2. When $y = z_2 g$, it can be derived that $x = cy + \frac{z_1 y}{z_2} = cz_2 g + z_1 g = c(d-1) + 2c = n > n-1$. Therefore, the possible values of y satisfying the equation $x = cy + \frac{z_1 y}{z_2}$ such that $0 \leqslant x \leqslant n-1$ are $0, z_2, 2z_2, \ldots, (g-1)z_2$. This further implies that $|L_0| = g = \gcd(2c, d-1)$. This completes the proof. □

Let $L_0 = \{x_0, x_1, \ldots, x_{m-1}\}$ and $L_{d-1} = \{y_0, y_1, \ldots, y_{m-1}\}$, where $x_i < x_{i+1}$ and $y_i < y_{i+1}$ for $0 \leqslant i < m-1$. Obviously, $x_0 = 0$ and $y_{m-1} = n-1$. The vertices in V can be partitioned into $V_0, \ldots, V_{m-1}$, where

$$V_i = \{\, x_i,\, x_i + 1,\, \ldots,\, y_i \,\} \quad i \in \{\, 0,\, \ldots,\, m-1 \,\}. \tag{2}$$

Example 1. Consider the case where $n = 16$ and $d = 3$ (see Figure 1). Then $O(1) = \{3,4,5\}$. We have $O_0(1) = 3$, $O_1(1) = 4$ and $O_2(1) = 5$. Notice that $O_0(0) = 0$ and $O_0(8) = 8$ and so $L_0(16,3) = \{0,8\}$. Similarly, $L_{d-1}(16,3) = L_2(16,3) = \{7,15\}$. This results in the partition $\{V_0, V_1\}$ of V, where $V_0 = \{0,1,\ldots,7\}$ and $V_1 = \{8,9,\ldots,15\}$.

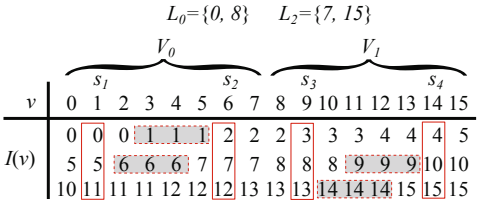


Fig. 1. A d-set in $G_B(16,3)$

Lemma 3. *Let* $L_0(n,d)$ *and* $L_{d-1}(n,d)$ *be defined as above. Let also* V_i *be defined as above, for* $0 \leqslant i \leqslant m-1$. *If* $d-1$ *is a multiple of* m, *then*

$$O_0(u_i) - u_i \equiv u_{y_p - i} - O_{d-1}(u_{y_p - i}) \pmod{n} \quad \textit{and}$$
$$O_{d-1}(u_i) - u_i \equiv u_{y_p - i} - O_0(u_{y_p - i}) \pmod{n}$$

where u_i *is the* i^{th} *element in* V_p *for* $0 \leqslant p \leqslant m-1$.

Proof. We prove this lemma for the case where $p = 0$. The other cases are similar. When $p = 0$, $u_i = i$. Then

$$O_0(u_i) - u_i = di - i = (d-1)i$$

and

$$\begin{aligned} u_{y_p - i} - O_{d-1}(u_{y_p - i}) &= (y_0 - i) - (d(y_0 - i) + d - 1) \\ &= (d-1)(-y_0 + i - 1) \\ &= (d-1)(-\frac{n}{m} + i) \\ &= -\frac{(d-1)n}{m} + (d-1)i \equiv (d-1)i \pmod{n} \end{aligned}$$

since m divides $d-1$. Here we use that $y_p = (p+1)\frac{n}{m} - 1$.

A similar argument shows that $O_{d-1}(u_i) - u_i \equiv u_{y_p - i} - O_0(u_{y_p - i}) \pmod{n}$. This completes the proof. □

Lemma 4. *Let* L_0, L_{d-1} *and* V_i *for* $0 \leqslant i \leqslant m-1$ *be as above. Assume that* $d-1$ *is a multiple of* m. *Let* $0 \leqslant p, q \leqslant m-1$ *and let* u_i *and* v_i *be the* i^{th} *element of* V_p *and* V_q. *Then*

(1) $O_0(u_i) - u_i \equiv O_0(v_i) - v_i \pmod{n}$, *and*
(2) $u_i - O_{d-1}(u_i) \equiv v_i - O_{d-1}(v_i) \pmod{n}$.

Proof. It suffices to show that $O_0(i) - i \equiv O_0(v_i) - v_i \pmod{n}$, i.e., $p = 0$. Via a similar argument as in Lemma 3 we derive

$$O_0(i) - i = di - i \quad \text{and} \quad O_0(v_i) - v_i = \frac{(d-1)pn}{m} + di - i.$$

Thus

$$O_0(u_i) - u_i \equiv O_0(v_i) - v_i \pmod{n}.$$

Similarly, $u_i - O_{d-1}(u_i) \equiv v_i - O_{d-1}(v_i) \pmod{n}$. This completes the proof. □

Definition 1. *Let* $c = \frac{n}{d+1}$. *A subset* $S = \{s_1, s_2, \ldots, s_c\}$ *of* V *with* $s_i < s_{i+1}$ *for* $1 \leqslant i \leqslant c-1$ *is a* d-set *if the following two conditions hold.*

(1) $s_{i+1} - s_i \geqslant d$ *for* $1 \leqslant i \leqslant c-1$ *and* $s_1 - s_c \pmod{n} \geqslant d$, *and*
(2) *for every* $s_i \in S$, $1 \leqslant i \leqslant c$, *there exists an* $s_j \in S$ *such that*

$$\textit{either} \quad O_0(s_i) \equiv s_j + 1 \pmod{n} \quad \textit{or} \quad O_{d-1}(s_i) \equiv s_j - 1 \pmod{n}.$$

Definition 2. *A d-set* $S = \{s_1, s_2, \ldots, s_c\}$ *is called a* one-to-one d-set, *abbreviated as od-set, if, for every* $s_i \in S$, *there exists a unique* $s_j \in S$ *such that*

$$O_0(s_i) \equiv s_j + 1 \pmod{n}.$$

We use the function σ to denote the mapping from i to j, i.e., $\sigma(i) = j$. Thus if S is an od-set, then $O_0(s_i) = s_{\sigma(i)} + 1$ and σ is a one-to-one function. For simplicity we also write $\sigma(x_1, x_2, \ldots, x_c) = (y_1, y_2, \ldots, y_c)$ if $\sigma(x_i) = y_i$ for $1 \leqslant i \leqslant c$.

Example 2. See Figure 1. We have $c = \frac{n}{d+1} = \frac{16}{4} = 4$. A d-set in $G_B(16,3)$ is $S = \{1,6,9,14\}$. It is easily to check that Condition (1) holds for $S = \{1,6,9,14\}$. To verify Condition (2), notice that

$$O_{d-1}(s_1) = O_2(1) \equiv d+2 \pmod{n} \equiv 5 \pmod{16} = 6 - 1 = s_2 - 1.$$

Similarly, $O_0(s_2) = s_1 + 1, O_{d-1}(s_3) = s_4 - 1$, and $O_0(s_4) = s_3 + 1$. That is, $\sigma(1,2,3,4) = (2,1,4,3)$.

To illustrate an od-set, we use $G_B(9,2)$ and $S = \{1,4,7\} = \{s_1, s_2, s_3\}$ as an example (see Figure 2). Clearly, Condition (1) holds for S. Notice that $O_0(1) = 2 = s_1 + 1, O_0(s_2) = O_0(4) = 8 = s_3 + 1$, and $O_0(s_3) = 5 = s_2 + 1$. That is, $\sigma(1,2,3) = (1,3,2)$.

		s_1			s_2			s_3	
v	0	1	2	3	4	5	6	7	8
$I(v)$	0	0	1	1	2	2	3	3	4
	4	5	5	6	6	7	7	8	8

Fig. 2. An od-set in $G_B(9,2)$

Lemma 5. *If there exists a d-set S in* $G_B(n,d)$*, then S is a perfect absorbant.*

Proof. Let $S = \{s_1, s_2, \ldots, s_c\}$ be a d-set of $G_B(n,d)$, where $c = \frac{n}{d+1}$. We show that S satisfies the Properties (i)-(iii) on Page 305 that define a perfect absorbant.

Let v be a vertex in $I(s_i)$ for some $1 \leqslant i \leqslant c$, i.e., $s_i \in O(v)$. Since $O(v)$ has the circular consecutive property, and $|O(v)| = d$, and since $s_{i+1} - s_i \geqslant d$ this implies that $v \notin I(s_j)$ for $j \neq i$ and therefore Property (i) holds.

By Condition (2) of a d-set, if $O_0(s_i) = s_j + 1$ or $O_{d-1}(s_i) = s_j - 1$, then $s_i \notin I(s_i)$ and $s_i \notin s_j$ for any $j \neq i$. Thus Properties (ii) and (iii) hold. This proves that S is a perfect absorbant. This completes the proof. □

Lemma 6. *If there exists an od-set S in* $G_B(n,d)$*, then S is also a perfect dominating set.*

Proof. By definition of od-sets, $O(s_i) \cap O(s_j) = \varnothing$. Furthermore,

$$|S \cup \bigcup_{s_i \in S} O(s_i)| = c(d+1).$$

Thus S is a perfect dominating set. □

Corollary 1. *If* $S = \{s_1, s_2, \ldots, s_c\}$ *is an od-set in* $G_B(n,d)$*, then*

$$s_{i+1} - s_i \equiv d+1 \pmod{n}$$

for $1 \leqslant i \leqslant c$*, where the indices are taken modulo* c*, that is,* $s_{c+1} = s_1$.

Theorem 3. *A set S is at the same time a perfect dominating set and a perfect absorbant of* $G_B(n,d)$ *if and only if S is an od-set.*

Proof. By Lemma 6, if S is an od-set then S is perfectly dominating as well as perfectly absorbant.

We prove the converse by means of contradiction. Let $S = \{s_1, s_2, \ldots, s_c\}$ be a perfect dominating set and a perfect absorbant. Assume that S is not an od-set. Since S is a perfect dominating set, $O(s_i) \cap O(s_j) = \varnothing$ for $i \neq j$. If $s_{i+1} - s_i = d+1$, then S is a perfect absorbant.

Assume that there exists an s_r, $1 \leqslant r \leqslant c$, such that $s_{r+1} - s_r \pmod{n} \leqslant d$ (with indices modulo c). For simplicity, assume that $1 \leqslant r \leqslant c-1$. The other case, that is, $r = c$, can be handled similarly. By the consecutive property of $O(s_i)$, if $x \in O(s_i)$ and $s_r < x < s_{r+1}$, then either s_r or s_{r+1} is also in $O(s_i)$. This implies that there exists at least one arc in the induced sub-digraph of S, a contradiction. This completes the proof. □

3 Perfect Absorbants of $G_B(n,d)$

To investigate the perfect absorbant of $G_B(n,d)$, we classify $G_B(n,d)$ into the following four classes. As usual, we let $c = \frac{n}{d+1}$ and we assume that $c > 1$ is a positive integer.

(1) $d = 2c(k-1)+2j$, for $k \geqslant 1$ and $0 \leqslant j \leqslant c-1$.
(2) $d = 2c(k-1)+2j+1$ for c even, $k \geqslant 1$, and $1 \leqslant j \leqslant c-1$.
(3) $d = 2c(k-1)+2j+1$ for c odd, $k \geqslant 1$, and $1 \leqslant j \leqslant c-1$.
(4) $d = 2c(k-1)+1$ for $k > 1$.

In this section, we shall show that there exists a d-set for any digraph in class (1) and a subclass in class (2). Moreover, we show that there exists an od-set for some digraphs in class (1). We leave it as an open problem to decide if there exist perfect absorbants in class (3). In Section 4 we show that there are no perfect absorbants in class (4) and in $G_B(6k, 2k-1)$.

Lemma 7. *Assume*

$$c \geqslant 2 \quad \textit{and} \quad d = 2c(k-1)+2j > 1 \quad \textit{for } k \geqslant 1 \textit{ and } 0 \leqslant j \leqslant c-1.$$

Then $G_B(n,d)$ *has a perfect absorbant.*

Proof. Let $S = \{s_1, s_2, \ldots, s_c\}$ be the set in which

$$s_i = \frac{n(i-1)}{c} + \frac{d}{2} \tag{3}$$

for $1 \leqslant i \leqslant c$ (see Figure 3 for an illustration). We show that S is a d-set of $G_B(n,d)$ in which $d = 2c(k-1)+2j > 1$ and $c \geqslant 2$ for $0 \leqslant j \leqslant c-1$ and $k \geqslant 1$.

Clearly,

$$s_{i+1} - s_i = \frac{ni}{c} + \frac{d}{2} - \left(\frac{n(i-1)}{c} + \frac{d}{2}\right) = \frac{n}{c} = d+1 \quad \text{for } 1 \leqslant i \leqslant c-1.$$

Notice also that

$$s_1 - s_c = -\frac{n(c-1)}{c} = -n + (d+1) \equiv d+1 \pmod{n}.$$

It remains to show that Condition (2) in Definition 1 is satisfied.

Let

$$\sigma(i) = (2i-1)\cdot j \pmod{c}. \tag{4}$$

We derive $O_0(s_i) = s_{\sigma(i)} + 1$ as follows.

$$
\begin{aligned}
O_0(s_i) &= d \cdot s_i \\
&= d \cdot \left(\frac{n(i-1)}{c} + \frac{d}{2}\right) \quad \text{(by Equation (3))} \\
&= \frac{2(c(k-1)+j)n(i-1)}{c} + \frac{d^2}{2} \quad \text{(since } d = 2c(k-1)+2j\text{)} \\
&\equiv \frac{2jn(i-1)}{c} + j(d+1) - \frac{d}{2} \pmod{n} \\
&\quad \left(\text{since } \frac{d(d+1)}{2} = (c(k-1)+j)(d+1) \equiv j(d+1) \pmod{n}\right) \\
&\equiv \frac{2ijn - jn - n}{c} - \frac{jn-n}{c} + j(d+1) - \frac{d}{2} \pmod{n} \\
&\equiv \frac{2ijn - jn - n}{c} - (j-1)(d+1) + j(d+1) - \frac{d}{2} \pmod{n} \\
&\equiv \frac{n((2i-1)j-1)}{c} + \frac{d}{2} + 1 \pmod{n} \\
&\equiv s_{(2i-1)j} + 1 \pmod{n} \quad \text{(by Equation (3))} \\
&= s_{\sigma(i)} + 1. \quad \text{(by Equation (4))}
\end{aligned}
$$

Therefore, S is a d-set. By Lemma 5, S is a perfect absorbant of $G_B(n,d)$ and $\gamma_a(G_B(n,d)) = c$. This completes the proof. □

Example 3. In Figure 3 we have $n = 12$, $c = 4$ and $d = 2$. In the equation $d = 2 = 2c(k-1)+2j$ we can take $k = j = 1$. Equation (4) yields $\sigma(i) = (2i-1)j \pmod{c} = (2i-1) \pmod{4}$ which gives $\sigma(1,2,3,4) = (1,3,1,3)$. Thus

(a) $O_0(s_1) = s_{\sigma(1)} + 1 = s_1 + 1 = 2$,
(b) $O_0(s_2) = s_{\sigma(2)} + 1 = s_3 + 1 = 8$,
(c) $O_0(s_3) = s_{\sigma(3)} + 1 = s_1 + 1 = 2$, and
(d) $O_0(s_4) = s_{\sigma(4)} + 1 = s_3 + 1 = 8$.

By Definition 1, $S = \{s_1, s_2, s_3, s_4\}$ is a d-set (but not an od-set).

		s_1			s_2			s_3			s_4	
v	0	1	2	3	4	5	6	7	8	9	10	11
$I(v)$	0	0	1	1	2	2	3	3	4	4	5	5
	6	6	7	7	8	8	9	9	10	10	11	11

Fig. 3. A d-set in $G_B(12,2)$

Lemma 8. *Assume that* $c \geqslant 3$ *is an odd integer and* $d = 2c(k-1)+2j$ *for* $k \geqslant 1$ *and* $1 \leqslant j < c$. *If* c *and* j *are relatively prime, then* $G_B(n,d)$ *has a perfect absorbant which is also a perfect dominating set.*

Proof. Let $S = \{s_1, s_2, \ldots, s_c\}$ be the set of vertices with

$$s_i = \frac{n(i-1)}{c} + \frac{d}{2}$$

for $1 \leqslant i \leqslant c$ (see Figure 2 for an illustration). We shall show that S is an od-set of $G_B(n,d)$.

By Lemma 7, S is a d-set. By Equation (4), we need to show that $\sigma(i) = (2i-1)j \pmod c$ is a one-to-one function. We claim that

$$\sigma(i) - \sigma(\ell) \equiv 2(i-\ell)\cdot j \equiv 0 \pmod c \quad \text{implies} \quad i = \ell.$$

Assume that $\sigma(i) - \sigma(\ell) \equiv 0 \pmod c$. Since c and $2j$ are relatively prime, $i-\ell$ has to be divided exactly by c. Since $|i-\ell| < c$, c does not divide $i-\ell$ unless $i = \ell$. This completes the proof. □

Example 4. In Figure 2, $c = 3$, $d = 2$, $k = 1$ and $j = 1$. Notice that $d = 2c(k-1)+2j = 2$. Define the map $\sigma(i) = (2i-1)j \pmod c$. Then σ is the permutation $\sigma(1,2,3) = (1,3,2)$. Indeed,

(i) $O_0(s_1) = s_{\sigma(1)} + 1 = s_1 + 1 = 2$,
(ii) $O_0(s_2) = s_{\sigma(2)} + 1 = s_3 + 1 = 8$, and
(iii) $O_0(s_3) = s_{\sigma(3)} + 1 = s_2 + 1 = 5$.

Lemma 9. *Assume that* $c \in \mathbb{N}$, $c \geqslant 2$ *and* c *even. Let* j *be an odd integer with* $1 \leqslant j < c$, $k \geqslant 2$ *and* $d = 2c(k-1)+2j+1$. *Then* $G_B(n,d)$ *has a perfect absorbant.*

Proof. Let $S = \{s_1, s_2, \ldots, s_c\}$ be the set of vertices of $G_B(n,d)$ with

$$s_i = (d+1)i - \frac{d+1}{2} - (i \pmod 2) \tag{5}$$

for $1 \leqslant i \leqslant c$ (see Figure 1 for an illustration).

Define

$$\sigma(i) = (2j+1)\cdot i - \frac{d-1}{2} \equiv (2j+1)\cdot i - j \pmod c. \tag{6}$$

To prove that S is a perfect absorbant of $G_B(n,d)$, we shall show that

(a) $O_{d-1}(s_i) = s_{\sigma(i)} - 1$ when i is odd, and
(b) $O_0(s_i) = s_{\sigma(i)} + 1$ when i is even.

Notice that $\sigma(i)$ is even if and only if i and j are the same parity, that is, both even or both odd.

First assume that i is odd. So, in that case, $\sigma(i)$ is even. Then we have

$$\begin{aligned}
O_{d-1}(s_i) &= d \cdot s_i + d - 1 \\
&= d \cdot ((d+1)i - \frac{d+1}{2} - 1) + d - 1 \quad \text{(by Equation (5))} \\
&= (2c(k-1)+2j+1)(d+1)i - \frac{d(d+1)}{2} - 1 \\
&\equiv (2j+1)(d+1)i - \frac{(d+1)(d-1)}{2} - \frac{d+1}{2} - 1 \pmod{n} \\
&= (d+1)((2j+1)i - \frac{d-1}{2}) - \frac{d+1}{2} - 1 \\
&= (d+1)\sigma(i) - \frac{d+1}{2} - 1 \quad \text{(by Equation (6))} \\
&= s_{\sigma(i)} - 1. \quad \text{(by Equation (5))}
\end{aligned}$$

By using a similar argument, we can derive $O_0(s_i) = s_{\sigma(i)} + 1$ when i is even. This completes the proof. □

Example 5. In Figure 1, since $c = 4$ and $d = 3$, this results in $k = 1$ and $j = 1$ in the equation $d = 2c(k-1) + 2j + 1 = 3$. By the mapping function $\sigma(i) = (2j+1)i - \frac{d-1}{2}$, $\sigma(1,2,3,4) = (2,1,4,3)$. Thus

(1) $O_{d-1}(s_1) = O_2(s_1) = s_{\sigma(1)} - 1 = s_2 - 1 = 5$,
(2) $O_0(s_2) = s_{\sigma(2)} + 1 = s_1 + 1 = 2$,
(3) $O_2(s_3) = s_{\sigma(3)} - 1 = s_4 - 1 = 13$, and
(4) $O_0(s_4) = s_{\sigma(4)} + 1 = s_3 + 1 = 10$.

For the case where c and j are even integers with $2 \leqslant j < c-2$ in $d = 2c(k-1)+2j+1$, there exist perfect absorbants for some digraphs in class (2), e.g., $G_B(24,5)$ in which $\{2,8,15,21\}$ is a perfect absorbant. However, there also exist some digraphs in class (2) which have no perfect absorbants, e.g., $G_B(36,5)$. We leave it as an open problem to find the subclass of class (2) in which all digraphs have perfect absorbants.

4 $G_B(n,d)$ with $d = 2c(k-1)+1$

Definition 3. *A vertex $x \notin I(x)$ is called Type I imperfect if $c+1 \leqslant O_0(x) - x \pmod n \leqslant d-1$ or $c+1 \leqslant x - O_{d-1}(x) \pmod n \leqslant d-1$.*

Definition 4. *A vertex x is called a far-neighbor if $O_0(x) - x \pmod n \geqslant d-1$ and $x - O_{d-1}(x) \pmod n \geqslant d-1$. A vertex x is called a near-neighbor if $0 < O_0(x) - x \pmod n \leqslant c$ or $0 < x - O_{d-1}(x) \pmod n \leqslant c$.*

For a far-neighbor x, let $F(x) = \{x-d+1, x-d+2, \ldots, x+d-1\} \cup O(x)$, where the numbers are taken modulo n. Hereafter, all numbers of vertices are modulo n unless stated otherwise. Consider a partition $\{F_1, F_2\}$ of $V \setminus F(x)$ into two sets

(a) $F_1(x) = \{x+d, x+d+1, \ldots, O_0(x)-1\}$, and
(b) $F_2(x) = \{O_{d-1}(x)+1, O_{d-1}(x)+2, \ldots, x-d\}$.

Definition 5. *A far-neighbor x is called Type II imperfect if there do not exist c_1 and c_2 with $c_1+c_2+1=c$ such that, for all $i \in \{1,2\}$, $(c_i-1)d+1 \leqslant |F_i(x)|$.*

For a near-neighbor x let $U(x) = \{x-d+1, x-d+2, \ldots, x+d-1\} \cup O(x)$. Let $\widehat{U}(x) = V \setminus U(x)$. Then

(1) $\widehat{U}(x) = \{O_{d-1}(x)+1, O_{d-1}(x)+2, \ldots, x-d\}$ if $0 < O_0(x)-x \pmod n \leqslant c$, and
(2) $\widehat{U}(x) = \{x+d, x+d+1, \ldots, O_0(x)-1\}$ if $0 < x-O_{d-1}(x) \pmod n \leqslant c$.

Definition 6. *A near-neighbor x is called Type III imperfect if the following condition is not satisfied.*

$$(c-2)d+1 \leqslant |V \setminus U(x)| \tag{7}$$

Notice that all Type I, II, and III imperfect vertices are not in any perfect absorbant.

Theorem 4. *Let $G_B(n,d)$ be the generalized De Bruijn graph with $n = c(d+1)$ and $d = 2c(k-1)+1$. If $c \geqslant 1$ and $k \geqslant 2$, then $\gamma_a(G_B(n,d)) = c+1$.*

Proof. Every nonself-loop vertex is either Type I, II, or III imperfect. This proves the theorem. □

Theorem 5. *Let $k \geqslant 2$. Then $\gamma_a(G_B(6k, 2k-1)) = 4$.*

Proof. All vertices x in V with $x \notin I(x)$ are Type I imperfect except the vertices in $H = \{3k-2, 3k-1, 6k-3, 6k-2\} \cup \{12k-5, 12k-4, 15k-6, 15k-5\}$ when $n = 18k-6$ and $d = 6k-3$. However, any two vertices u and v in the same set, i.e., $\{3k-2, 3k-1, 6k-3, 6k-2\}$ or $\{12k-5, 12k-4, 15k-6, 15k-5\}$ results in $I(u) \cap I(v) \neq \varnothing$. This completes the proof. □

5 Concluding Remarks

In this paper, we investigate the perfect absorbant problem in generalized De Bruijn digraphs. We have shown that there exist perfect absorbants for class (1) and a subclass of class (2). Furthermore, we also show that a digraph in a subclass of class (1) has $\gamma(G_B(n,d)) = \gamma_a(G_B(n,d)) = c$. We also show that there does not exist any perfect absorbant in the digraphs in class (4). For digraphs in class (3), we only show that there does not exist any perfect absorbant when $c = 3$. Note that $G_B(8k-4, 4k-3)$ is in class (4). Thus, by Theorems 2 and 4, $\gamma_a(G_B(8k-4, 4k-3)) = 3$. By Theorem 5, $\gamma_a(G_B(6k, 2k-1)) = 4$. This affirms the two conjectures in [10]. Some special subclasses of class (3) also have no perfect absorbant, e.g., $c = d$. As a future study, we have the following conjectures:

1. There is a perfect absorbant in $G_B(c(d+1), d)$ if and only if c is a multiple of $|L_0|$.
2. There exists a perfect absorbant which is also a perfect dominating set if and only if $d = 2c(k-1)+2j$, for $1 \leqslant j < c$, and odd $c \geqslant 3$ is relatively prime to j.

References

1. Bermond, J.-C., Peyrat, C.: De Bruijn and Kautz networks: a competitor for the hypercube? In: Andrè, F., Verjus, J.P. (eds.) Hypercube and Distributed Computers, pp. 279–293. Elsevier Science Publishers B.V, North-Holland (1989)
2. Du, D.Z., Hsu, D.F., Hwang, F.K.: Doubly-linked ring networks. IEEE Transactions on Computputers C-34, 1025–1035 (1985)
3. Du, D.Z., Wang, F.K.: Generalized De Bruijn digraphs. Networks 18, 27–38 (1988)
4. Esfahanian, A.H., Hakimi, S.L.: Fault-tolerant routing in DeBruijn communication networks. IEEE Transactions on Computputers C-34, 777–788 (1985)
5. Imase, M., Itoh, M.: Design to minimize diameter on buildingblock network. IEEE Transactions on Computputers C-30, 439–442 (1981)
6. Imase, M., Itoh, M.: A design for directed graphs with minimum diameter. IEEE Transactions on Computputers C-32, 782–784 (1983)
7. Imase, M., Soneoka, T., Okada, K.: Connectivity of regular directed graphs with small diameters. IEEE Transactions on Computputers C-34, 267–273 (1985)
8. Kikuchi, Y., Shibata, Y.: On the domination numbers of generalized de Bruijn digraphs and generalized Kautz digraphs. Information Processing Letters 86, 79–85 (2003)
9. Reddy, S.M., Pradhan, D.K., Kuhl, J.G.: Direct graphs with minimum diameter and maximal connectivity. School of Engineering, Oakland University Tech. Rep. (July 1980)
10. Shan, E., Cheng, T.C.E., Kang, L.: Absorbant of Generalized De Bruijn Digraphs. Information Processing Letters 105, 6–11 (2007)

Multi-Multiway Cut Problem on Graphs of Bounded Branch Width

Xiaojie Deng, Bingkai Lin, and Chihao Zhang

Department of Computer Science, Shanghai Jiao Tong University
{lvchaxj,kai314159,chihao.zhang}@gmail.com

Abstract. The Multi-Multiway Cut problem proposed by Avidor and Langberg[2] is a natural generalization of Multicut and Multiway Cut problems. That is, given a simple graph G and c sets of vertices $S_1, \cdots, S_c$, the problem asks for a minimum set of edges whose removal disconnects every pair of vertices in S_i for all $1 \leq i \leq c$. In [13], the authors asked whether the problem is polynomial time solvable for fixed c on trees. In this paper, we give both a logical approach and a dynamic programming approach to the Multi-Multiway Cut problem on graphs of bounded branch width, which is exactly the class of graphs with bounded treewidth. In fact, for fixed c and branch width k, we show that the Multi-Multiway Cut problem can be solved in linear time, thus affirmatively answer the question in [13].

1 Introduction

The Multi-Multiway Cut problem is defined as follows[2]:

Multi-Multiway Cut($\mathcal{G}$)
Input: A simple graph $G = (V, E) \in \mathcal{G}$ and c sets of vertices $S_1, S_2, \ldots, S_c$.
Problem: Find a set of edges $C \subseteq E$ with minimum cardinality whose removal disconnects every pair of vertices in each set S_i

The well-studied Multiway Cut problem is a special case of Multi-Multiway Cut when $c = 1$ and Multicut problem is a special case of Multi-Multiway Cut when every S_i contains exactly two vertices. Multiway Cut is NP-hard on general graphs [7] and Multicut is NP-hard even on trees (by a simple reduction from vertex cover to stars). Based on these two hardness results, [13] asked whether Multi-Multiway Cut is polynomial-time solvable on trees when c is a constant.

In this paper, we show that when $\mathcal{G}$ is the class of graphs of branch width at most k, Multi-Multiway Cut($\mathcal{G}$) can be solved in $O(k^{2k+2} \cdot 2^{2kc} \cdot |G|)$. The notion of branch width is equivalent to treewidth up to constant, which roughly measures how similar a graph is to a tree. We will present two algorithms.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 315–324, 2013.

The first one is using logical approach based on Courcelle's theorem, i.e., we will give a monadic second order formula characterizing Multi-Multiway Cut and Courcelle's theorem directly implies a linear algorithm for fixed c and k. However, the canonical algorithm behind Courcelle's theorem contains huge hidden constant and thus it is impractical. Our second algorithm is based on dynamic programming and can be viewed as a subtle way of applying Courcelle's theorem to specific problem, thus it is more efficient.

Related Work. The Multicut problem is known to be APX-hard for $c \geq 3$ [7]. An $O(\log c)$-approximation algorithm based on the well-known region growing rounding technique was presented in [8]. From the parameterized complexity point of view, various parameters have been studied like solution size [4], cardinality k and solution size [14] [17], or treewidth of the input structure [9],[15].

The Multiway Cut problem is also known to be APX-hard for $\ell \geq 3$ [7], where $\ell = |S_1|$. A $(3/2 - 1/\ell)$-approximation algorithm was presented in [5]. Later Karger et al. [12] improved the approximation ratio to $(1.3438 - \varepsilon_m)$. In terms of exact algorithms, [17] gave an algorithm in $O(2^{(\frac{\ell-2}{\ell-1}k)}\ell T(n,m))$ time, where k is the solution size, $T(n,m) = O(\min(n^{2/3}, m^{1/2})m)$, n is the number of vertices and m is the number of edges in graph.

For Multi-Multiway Cut problem, Avidor and Langberg [2] presented an $O(\log k)$-approximation algorithm using the idea of region growing. [18] studied some variant of Multi-Multiway Cut on trees including the prize-collecting version.

In [13], Liu and Zhang gave a fixed-parameter tractable algorithm for Multi-Multiway Cut on trees in which the parameter is the size of solution and c is constant. They left the question whether the problem is polynomially solvable as an open problem. We answer this question affirmatively in this paper, as our result holds for graphs of bounded branchwidth (certainly for trees). Independently, Kanj et al. showed that Multi-Multiway Cut on trees is polynomially solvable for a constant c [11]. Their result only holds for trees but the running time is better than ours.

Our Result. We present two linear algorithms for Multi-Multiway Cut with constant c and k, where k is the branch width of input graph. Our algorithms can be further extended to graphs with weight on edges and directed graph whose underlying undirected graph is of bounded branch width.

Organization of the Paper. In Section 2, we introduce some necessary background and notations. In Section 3, we use Courcelle's Theorem to give an algorithm for Multi-Multiway Cut on graphs of bounded branch width. Next, in Section 4, we present the dynamic programming algorithm and finally conclude the paper in Secion 5.

2 Preliminaries

$\mathbb{N}$ denote the set of natural numbers. For a set S, the set of all k-element subsets of S is $[S]^k$. The power set of S is denoted by $\mathcal{P}(S) = \{X | X \subseteq S\}$. $|S|$ is the cardinality of S.

2.1 Graph

A graph is a pair $G = (V, E)$, where V is a finite set of vertices and $E \subseteq V^2$. The size of G is $|G| = |V| + |E|$. An edge $\{u, v\}$ is also written as uv. We also denote the edges set and vertices set of G as $E(G)$ and $V(G)$. A graph with labels is $G = (V, E, L_1, L_2, \cdots, L_l)$, where each $L_i \subseteq V$ is a label set. The union of two graph G and H is $G \cup H = (V(G) \cup V(H), E(G) \cup E(H))$.

2.2 Branch Decomposition

Definition 1 (Branch Decomposition). *Given a graph $G = (V, E)$, a branch decomposition is a pair (T, β), such that*

1 T is a binary tree with $|E|$ leaves and every inner node of T has two children.
2 β is a mapping from $V(T)$ to $\mathcal{P}(E)$ satisfying the following conditions:
  2.1 For each leaf $v \in V(T)$, there exists $e \in E(G)$ with $\beta(v) = \{e\}$, and there are no $v, u \in V(T), v \neq u$ such that $\beta(v) = \beta(u)$.
  2.2 For every inner node $v \in V(T)$ with children v_l, v_r, $\beta(v) = \beta(v_l) \cup \beta(v_r)$;

Definition 2 (Boundary). *Given a graph $G = (V, E)$, for every set $F \subseteq E$, the boundary $\partial F = \{v \mid v$ is incident to edges in F and $E \setminus F\}$.*

Definition 3 (Width of a Branch Decomposition). *Given a branch decomposition (T, β) of $G = (V, E)$, the width of this decomposition is* $\max\{|\partial\beta(v)| \mid v \in V(T)\}$.

The branch width $bw(G)$ of G is defined as the minimum width of all branch decompositions (T, β) for G.

Proposition 1. *[3] For any fixed k, there is a linear time algorithm that checks if a graph has branch width k and, if so, outputs a branch decomposition of minimum width.*

Branch width is related to another well-known graph parameter, treewidth. Indeed, they are equivalent up to constant. Let $tw(G)$ be the treewidth of graph G, then

Proposition 2. *[16]* $bw(G) \leq tw(G) + 1 \leq \max\{\frac{3}{2} \cdot bw(G), 2\}$.

Definition 4. *Given a branch decomposition (T, β) of $G = (V, E)$, for any $t \in V(T)$, let $\mathcal{V}_t = \{v \mid \exists u \in V(G), vu \in \beta(t)\}$, the subgraph $\mathcal{G}_t$ is*

$$\mathcal{G}_t = (\mathcal{V}_t, \beta(t))$$

.

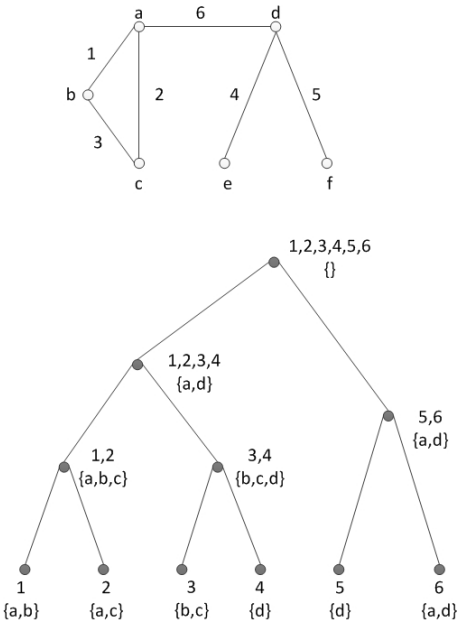


Fig. 1. A graph with a branch decomposition of width 3

2.3 Logic

We use FO and MSO to denote First Order Logic and Monadic Second Order Logic respectively. The difference between these two logic is that FO only allows quantification over individual variables while MSO allows quantification over set variables. Furthermore, MSO can be extended to MSO_2. In this paper, MSO_2 is the logic that allows quantification over subset of edges in graph.

3 Logical Approach

We consider a Multi-Multiway Cut instance $(G, S_1, \cdots, S_c)$ where $G = (V, E)$ is a simple graph of branch width k. Many NP-hard problems on graphs have efficient solutions when the input graphs have bounded treewidth by the following theorem:

Theorem 1 (Courcelle's Theorem, Optimization Version). *[6,1] Given an* MSO *formula $\phi(U)$, there is an algorithm A satisfies that for any labeled graph $G = (V, E, L_1, \cdots, L_l)$ of treewidth k, A computes the set $U \subseteq V$ with minimum cardinality, such that $G \vDash \phi(U)$, with running time bounded by $f(k, |\phi(U)|)|G|$ for some computable function f.*

By Proposition 2, the branch width of a graph is equivalent to its treewidth up to some constant. Therefore Theorem 1 holds for graph with bounded branch width.

Furthermore, the optimization version of Courcelle's Theorem for MSO can be extend to MSO_2 formula by turning the original graph G into a labeled graph $I(G) = (V_I, E_I, L_V, L_E)$, where $V_I = V(G) \cup E(G)$, $L_V = V(G), L_E = E(G)$ and $E_I = \{\{v, e\} \mid v \in e\}$. Any MSO_2 formula about the original graph G can be translated into an MSO formula about $I(G)$. Since the graph $I(G)$ is also of bounded branch width, we have an optimization version of Courcelle's theorem for MSO_2. Readers can refer [10] for more detail.

Thus it is sufficient to write down an MSO_2 formula capturing Multi-Multiway Cut. We begin with a MSO_2 formula saying two vertices x, y are connected without edges in C:

$$\mathsf{conn}(x, y, C) = \forall X[X(x) \wedge (\forall u \forall v X(u) \wedge E(u, v) \wedge \neg C(u, v) \rightarrow X(v)) \rightarrow X(y)]$$

The Multi-Multiway Cut problem can be captured by an MSO_2 formula

$$\mathsf{mmcut}(C) = \bigwedge_{i=1}^{c} \forall x \forall y (S_i(x) \wedge S_i(y) \rightarrow \neg \mathsf{conn}(x, y, C))$$

Therefore for all $C \subseteq E(G), G \vDash \mathsf{mmcut}(C) \Leftrightarrow C$ is a multi-multiway cut of $(G, S_1, \cdots, S_c)$, thus we can solve Multi-Multiway Cut on bounded branch width graph via Courcelle's Theorem in time $f(k, c) \cdot |G|$ for some computable function f.

4 Dynamic Programming Approach

Let $(G, S_1, \ldots, S_c)$ be an instance of Multi-Multiway Cut and (T, β) be a branch decomposition of G of width k. It is convenient to view each $i \in [c]$ as a color, i.e., $v \in S_i$ means v is assigned color i. Note that a vertex v may be assigned with multiple colors (or no color) since we do not require $S_i \cap S_j = \varnothing$ for $i \neq j$. We use $col(v)$ to denote the set of colors v assigned. Thus a multi-multiway cut is an edge set whose removal disconnects all pairs of vertices with common color.

In fact, we will compute a table $C(t, Z)$ for every $t \in V(T)$ and Z. Here $Z = \{(X_i, Y_i) \mid i \in I\}$ for some index set I. $\{X_i \mid i \in I\}$ is a partition of $\partial\beta(t)$ and each Y_i is a set of colors. We say a set of edges $C \subseteq \beta(t)$ is *consistent with* Z if and only if

(1) C is a multi-multiway cut of $\mathcal{G}_t$.
(2) Let $H = (\mathcal{V}_t, \beta(t) \setminus C)$, i.e., the subgraph of $\mathcal{G}_t$ after removing C. Let $\{P_i \mid i \in I\}$ be the family of connected components in H such that $P_i \cap \partial\beta(t) \neq \varnothing$ for all $i \in I$. Then $X_i = P_i \cap \partial\beta(t)$ and $Y_i = \bigcup_{v \in P_i} col(v)$ for all $i \in I$.

It is easy to see that, for every multi-multiway cut C of $\mathcal{G}_t$, there is only one consistent Z. Intuitively, Z encodes colors exposed to external when removing C from $\mathcal{G}_t$.

The value of $C(t, Z)$ is a minimum edge set $C \subseteq \beta(t)$ that is consistent with Z, if there are more than one C consistent with Z with minimum cardinality, then $C(t, Z)$ is arbitrary one of them. If no such C exists, then the value of $C(t, Z)$ is "Impossible". Indeed, then minimum multi-multiway cut of G is the one in $\{C(r, Z) \mid \text{ all possible } Z\}$ with minimum cardinality, where r is the root of T.

In the following, we will show how to compute $C(t, Z)$ recursively.

4.1 Computing $C(t, Z)$

If t is a leaf in T, the computation of $C(t, Z)$ is easy, otherwise let t_ℓ and t_r be its two children in T. $C(t, Z)$ is computed from some $C(t_\ell, Z_\ell)$ and $C(t_r, Z_r)$. We will use the following algorithm as a subroutine:

Merge Two Cuts

Input: (C_ℓ, Z_ℓ) and (C_r, Z_r), where Z_ℓ(resp. Z_r) is consistent with C_ℓ(resp. C_r)

Output: If $C_\ell \cup C_r$ is a multi-multiway cut of $\mathcal{G}_t$, compute the set Z consistent with $C_\ell \cup C_r$ on $\mathcal{G}_t$, otherwise return "Not Mergeable"

1 Let $Z_\ell = \{(X_i, Y_i) \mid i \in I\}, Z_r = \{(X_j, Y_j) \mid j \in J\}$ where I and J are two index sets.
2 Construct a bipartite graph $B = (I, J, E)$, for every $i \in I$ and $j \in J$, $ij \in E$ if and only if $X_i \cap X_j \neq \varnothing$.
3 Let $\{P_s \mid s \in S\}$ be family of connected components in B where S is an index set.
4 For every $s \in S$ and $p, q \in P_s$, if $Y_p \cap Y_q \neq \bigcup_{v \in X_p \cap X_q} col(v)$, return "Not Mergeable".
5 For $s \in S$:
  5.1 Let $X_s = \left(\bigcup_{p \in P_s} X_p\right) \cap \beta(t)$
  5.2 Let $Y_s = \left(\bigcup_{p \in P_s} Y_p\right)$
6 return $Z = \{(X_s, Y_s) \mid s \in S \text{ and } X_s \neq \varnothing\}$.

Algorithm 1: Merge Two Cuts

Step 4 in Algorithm 1 checks whether there are two vertices sharing some common color being connected after merging. If it is the case, the algorithm outputs "Not Mergeable", which means $C_\ell \cup C_r$ is not a multi-multiway cut of G. The size of bipartite graph B is $O(k^2)$ and all other operations are linear to the size of B, so this algorithm is in $O(k^2)$.

Fig 2 shows an example where $X_{u_1} \cap X_{v_1} = \{1\}$, $X_{u_3} \cap X_{v_2} = \{2\}$ and $X_{u_3} \cap X_{v_4} = \{3\}$; with the corresponding color sets $Y_{u_1} = \{white, black\}$, $Y_{u_3} = \{white, black, gray\}$, $Y_{v_1} = \{black, gray\}$ and $Y_{v_2} = \{white, black, gray\}$. Applying Algorithm 1 to this example, in the Step 4 of Algorithm 1, we can find u_3, v_2

in the same component of the bipartite graph, and $Y_{u_3} \cap Y_{v_2} = \{gray, white\}$, while $\bigcup_{v \in X_{u_3} \cap X_{v_2}} col(v) = \{gray\}$. The algorithm will return "Not Mergeable", since there are two vertices with the common color white being connected after merging.

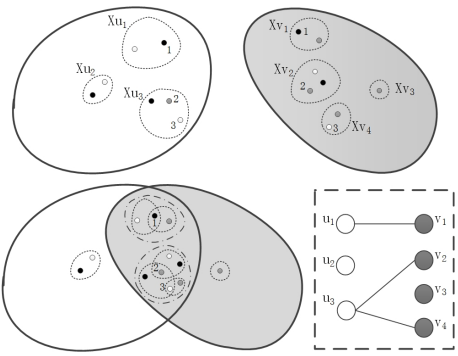


Fig. 2. Merge Two Pairs

To ease the presentation, we define a binary operation $\oplus$:

Definition 5. *$Z_\ell \oplus Z_r = Z$ if and only if Algorithm 1 output Z on inputs Z_ℓ and Z_r.*

Compute $C(t, Z)$

Input: $(G, S_1, \cdots, S_c)$ along with a branch decomposition (T, β) of width k

Output: Compute the table $C(t, Z)$

1 If t is a leaf in T. Let uv be the unique edge in $\beta(t)$. There are two cases:

(1.1) $Z = \{(X_1, Y_1)\}$ where $X_1 = \{u, v\}$. In this case $C(t, Z) = \varnothing$ if $col(u) \cap col(v) = \varnothing$ and $Y_1 = col(u) \cup col(v)$. Otherwise, $C(t, Z) =$"Impossible";

(1.2) $Z = \{(X_1, Y_1), (X_2, Y_2)\}$ where $X_1 = \{u\}$ and $X_2 = \{v\}$. In this case $C(t, Z) = \{uv\}$ if $Y_1 = col(u), Y_2 = col(v)$. Otherwise, $C(t, Z) =$"Impossible".

2 If t is not a leaf in T, let t_ℓ and t_r be its two children.

$$C(t, Z) = \arg\min_{|C|}\{C \mid C = C(t_\ell, Z_\ell) \cup C(t_r, Z_r) \text{ and } Z = Z_\ell \oplus Z_r\}$$

Algorithm 2: Compute $C(t, Z)$

We can now compute $C(t, Z)$ as follows: $C(t, Z)$ is the minimum edge set $C \subseteq \beta(t)$ such that $C = C(t_\ell, Z_\ell) \cup C(t_r, Z_r)$ for some Z_ℓ, Z_r satisfying $Z_\ell \oplus Z_r = Z$.

We enumerate all such Z_ℓ and Z_r to compute $C(t, Z)$. Note that if one side of the union is "Impossible", then the result is also "Impossible". In summary, we use Algorithm 2 to compute $C(t, Z)$.

Putting all together, we have:

Theorem 2. *Given a* Multi-Multiway Cut *instance* $(G, S_1, \cdots, S_c)$ *along with a branch decomposition* (T, β) *of width* k*, the minimum multi-multiway cut can be computed in time* $O(k^{2k+2} \cdot 2^{2kc} \cdot |G|)$.

Proof. First note that, for a fixed k, the number of distinct Z is at most $k^k 2^{ck}$. This is because

(1) $|\beta(t)| \leq k$ and the number of partitions of a k-size set is upper bounded by k^k, and
(2) There are at most 2^c distinct sets of colors and each Y_i is one of them.

In Algorithm 2, $C(t, Z)$ is computed in a bottom-up style: once we know all $C(t_\ell, Z_\ell)$ and $C(t_r, Z_r)$, we can enumerate all of them to compute the corresponding $C(t, Z)$. We need to deal with at most $(k^k 2^{ck})^2$ pairs of Z_ℓ and Z_r and for each pair, we use Algorithm 1 to compute their merging results, and then take the minimum $C(t, Z)$ for each Z. After computing the table $C(t, Z)$, we choose the cut in $\{C(r, Z) \mid \text{ all possible } Z\}$ with minimum cardinality as the final answer, where r is the root of T. In all, we use $O(k^{2k+2} \cdot 2^{2kc} \cdot |G|)$ time. □

4.2 Proof of the Correctness

The correctness of Algorithm 1

Lemma 1. *If* C_ℓ *is a multi-multiway cut consistent with* Z_ℓ *on* $\mathcal{G}_{t_\ell}$*,* C_r *is a multi-multiway cut consistent with* Z_r *on* $\mathcal{G}_{t_r}$ *and* $Z_\ell \oplus Z_r = Z$*, then* $C := C_\ell \cup C_r$ *is a multi-multiway cut consistent with* Z *on* $\mathcal{G}_t$.

Proof. Assume $Z = \{(X_k, Y_k) \mid k \in K\}$ for index set K.

First, we need to show that if $Z_\ell \oplus Z_r = Z$, then C is a multi-multiway cut of $\mathcal{G}_t$. That is, there are no two vertices with common color connected in $\mathcal{G}_t$ after removing C. Let $\mathcal{P}_t = \{P_s \mid s \in S\}$ be the family of connected components in $\mathcal{G}_t$ after removing C, $\mathcal{P}_\ell = \{P_i \mid i \in I\}$ (resp. $\mathcal{P}_r = \{P_j \mid j \in J\}$) be the set of connected components in $\mathcal{G}_{t_\ell}$ (resp. $\mathcal{G}_{t_r}$) after removing C_ℓ (resp. C_r). It follows from the definition of branch decomposition that

$$P_s = \bigcup_{i \in I'} P_i \cup \bigcup_{j \in J'} P_j$$

for some index sets $I' \subseteq I$ and $J' \subseteq J$.

Thus if P_s contains two vertices with common color, Step 4 of Algorithm 1 will return "Not Mergeable", but this is impossible since $Z_\ell \oplus Z_r = Z$ means Algorithm 1 outputs Z on inputs Z_ℓ, Z_r.

It remains to verify $\{X_k \mid k \in K\}$ is exactly $\{P_s \cap \partial\beta(t) \mid s \in S \text{ and } P_s \cap \partial\beta(t) \neq \varnothing\}$ and Y_k is the set of colors appeared in the component containing X_k.

Let P_s be a component in $\mathcal{P}_t$ such that $P_s \cap \partial\beta(t) \neq \varnothing$. It follows from the definition of branch decomposition that

$$P_s = \bigcup_{i \in I'} P_i \cup \bigcup_{j \in J'} P_j$$

for some index sets $I' \subseteq I$ and $J' \subseteq J$.

Then $I' \cup J'$ is a connected component in the bipartite graph B constructed in Step 2. The corresponding $X_k := \left(\bigcup_{s \in I' \cup J'} X_s\right) \cap \beta(t)$. It is easy to verify that the correspondence is a bijection between two sets.

The consistency of $\{Y_k \mid k \in K\}$ follows from Step 5.2 of Algorithm 1 and consistency of $\{X_k \mid k \in K\}$ directly. □

Now we prove the correctness of the Algorithm 2.

Lemma 2. *$C(t, Z)$ computed in Algorithm 2 is the minimum cut that consistent with Z on $\mathcal{G}_t$.*

Proof. We apply induction on the branch decomposition tree.

If t is leaf, the correctness is obvious. Otherwise, t has two children t_ℓ and t_r in T. Suppose C is the minimum cut that consistent with Z on $\mathcal{G}_t$, Let $C_\ell = C \cap \beta(t_\ell)$ and $C_r = C \cap \beta(t_r)$. There is only one Z_ℓ (resp. Z_r) that C_ℓ (resp. Z_r) is consistent with. Since $C = C_\ell \cup C_r$, we have $Z_r \oplus Z_\ell = Z$ by the Algorithm 1.

Let $C'_\ell = C(t_\ell, Z_\ell)$, we have $|C'_\ell| \leq |C_\ell|$ by the induction; similarly let $C'_r = C(t_r, Z_r)$, we have $|C'_r| \leq |C_r|$. Thus $C'_\ell \cup C'_r$ is a cut consistent with $Z_\ell \oplus Z_r$. On the other hand, $C_\ell \cup C_r$ is also a cut consistent with $Z_\ell \oplus Z_r$. We have $|C'_\ell \cup C'_r| \leq |C_\ell \cup C_r| = |C| \leq |C'_\ell \cup C'_r|$. According to our algorithm, $C(t, Z)$ is a cut of minimum cardinality over all $C(t_\ell, Z_\ell) \cup C(t_r, Z_r)$ for $Z_\ell \oplus Z_r = Z$, therefore $|C(t, Z)| = |C|$. □

5 Conclusion

In this paper, we presented two linear algorithms for Multi-Multiway Cut problem with constant number of terminal sets on graphs of bounded branch width. The logical approach is straightforward and very easy to design, however, it relies on a canonical algorithm and thus not practical. Our second approach is somehow a refinement of the canonical algorithm to specific problem, the more subtle design of subproblem gains much efficiency.

Acknowledgements. The work is supported by NSF of China (61033002) and by Science and Technology Commission of Shanghai Municipality (11XD1402800).

References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. Journal of Algorithms 12(2), 308–340 (1991)
2. Avidor, A., Langberg, M.: The multi-multiway cut problem. Theoretical Computer Science 377(1-3), 35–42 (2007)
3. Bodlaender, H.L., Thilikos, D.M.: Constructive linear time algorithms for branchwidth. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 627–637. Springer, Heidelberg (1997)
4. Bousquet, N., Daligault, J., Thomassé, S., Yeo, A., et al: A polynomial kernel for multicut in trees. In: 26th International Symposium on Theoretical Aspects of Computer Science STACS 2009, pp. 183–194 (2009)
5. Călinescu, G., Karloff, H., Rabani, Y.: An improved approximation algorithm for multiway cut. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pp. 48–52. ACM (1998)
6. Courcelle, B.: Graph rewriting: An algebraic and logic approach. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), pp. 193–242. Elsevier and MIT Press (1990)
7. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The complexity of multiterminal cuts. SIAM Journal on Computing 23(4), 864–894 (1994)
8. Garg, N., Vazirani, V.V., Yannakakis, M.: Approximate max-flow min-(multi) cut theorems and their applications. SIAM Journal on Computing 25(2), 235–251 (1996)
9. Gottlob, G., Lee, S.T.: A logical approach to multicut problems. Information Processing Letters 103(4), 136–141 (2007)
10. Grohe, M.: Logic, graphs, and algorithms. Logic and Automata–History and Perspectives, 357–422 (2007)
11. Kanj, I., Lin, G., Liu, T., Tong, W., Xia, G., Xu, J., Yang, B., Zhang, F., Zhang, P., Zhu, B.: Algorithms for cut problems on trees. arXiv:1304.3635 (2013)
12. Karger, D.R., Klein, P., Stein, C., Thorup, M., Young, N.E.: Rounding algorithms for a geometric embedding of minimum multiway cut. Mathematics of Operations Research 29(3), 436–461 (2004)
13. Liu, H., Zhang, P.: On the generalized multiway cut in trees problem. Journal of Combinatorial Optimization, 1–13 (2012)
14. Marx, D.: Parameterized graph separation problems. Theoretical Computer Science 351(3), 394–406 (2006)
15. Pichler, R., Rümmele, S., Woltran, S.: Multicut algorithms via tree decompositions. In: Calamoneri, T., Diaz, J. (eds.) CIAC 2010. LNCS, vol. 6078, pp. 167–179. Springer, Heidelberg (2010)
16. Robertson, N., Seymour, P.D.: Graph minors. x. obstructions to tree-decomposition. Journal of Combinatorial Theory, Series B 52(2), 153–190 (1991)
17. Xiao, M.: Simple and improved parameterized algorithms for multiterminal cuts. Theory of Computing Systems 46(4), 723–736 (2010)
18. Zhang, P.: Approximating generalized multicut on trees. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 799–808. Springer, Heidelberg (2007)

Bi-criteria Scheduling on Multiple Machines Subject to Machine Availability Constraints

Yumei Huo[1,⋆] and Hairong Zhao[2]

[1] Department of Computer Science,
College of Staten Island, CUNY, Staten Island, New York 10314, USA
yumei.huo@csi.cuny.edu
[2] Department of Mathematics, Computer Science & Statistics,
Purdue University Calumet, Hammond, IN 46323, USA
hairong@purduecal.edu

Abstract. This paper studies bi-criteria scheduling problems on m parallel machines with machine unavailable intervals. The goal is to minimize the total completion time subject to the constraint that the makespan is at most a constant T. We study two different unavailability models. In the first model, each machine has a single unavailable interval which starts from time 0. In the second model, each machine can have multiple unavailable intervals, but at any time, there is at most one machine unavailable. For each model, we show that there is an optimal polynomial time algorithm.

1 Introduction

Multi-criteria scheduling and scheduling subject to machine availability constraints have been two very active areas in manufacturing and operations management research over the last couple of decades. However, most research in these two areas has been conducted independently from one another. The only work that concerns both bi-criteria scheduling and scheduling with limited machine availability simultaneously is done by the authors in paper [6]. The paper [6] considers preemptively scheduling the jobs on two machines to minimize the total completion time and makespan at the same time with one as primary criterion and the other as secondary criterion.

In this paper, we continue our research to m, $m > 2$, parallel machines with unavailability constraint. We focus on preemptive schedules. A job can be preempted by another job or interrupted by machine unavailable intervals and resumed later on any available machine. We are concerned with the makespan and total completion time. The research in this paper is motivated not only by the lack of research results in this area, but also by its important applications in reality. The makespan and total completion time are two objectives of considerable interest. Minimizing makespan can ensure a good balance of the load among the machines and minimizing the total completion time can minimize the inventory

⋆ This work is supported by PSC-CUNY Research Award.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 325–338, 2013.

holding costs. It is quite common that the manufacturers wish to minimize both objectives.

On the other hand, each machine may have some unavailable periods during which it can not process any jobs. We study two different machine unavailability models: (1) Each machine has only one unavailable period which starts at time zero, that is, machine M_i $(1 \leq i \leq m)$ is available in interval $[r_i, \infty)$, where $r_i \geq 0$. We say in this case that each machine has a release time; (2) Each machine may have multiple unavailable periods, but there is at most one machine unavailable at any time. In reality, the first machine unavailability model may occur due to the unfinished jobs in the previous scheduling horizon and the second machine model exists in many scenarios since the preventive maintenance or periodical repair is usually done on a rotation basis instead of maintaining or repairing several machines simultaneously.

Formally speaking, there are n jobs that need to be scheduled on m machines, denoted by $J = \{J_1, J_2, \cdots, J_n\}$. Each job J_j has a processing time p_j. Without loss of generality, the processing times of jobs are assumed to be integer. Let S be a feasible schedule of these jobs, the completion time of job J_j in schedule S is denoted by $C_j(S)$. If S is clear from the context, we will use C_j for short. The makespan of S is $C_{\max}(S) = \max\{C_j(S)\}$, and the total completion time of S is $\sum C_j(S)$. We will use $C^*_{\max}$ to denote the minimum makespan. The goal is to schedule the set of jobs on m parallel machines so as to minimize $\sum C_i$ subject to the machine unavailability constraint and the condition that $C_{\max} \leq T$, where $T \geq C^*_{\max}$. To denote our problems, we extend the 3-field notation $\alpha \mid \beta \mid \gamma$ introduced by Graham et al. [3]. For the model that each machine has a release time, our problem is denoted by $P_m, r_i \mid r - a, prmt \mid \sum C_j / C_{\max} < T$. For the model that there is at most one machine unavailable at any time, our problem is denoted by $P_{m-1,1} \mid r - a, prmt \mid \sum C_j / C_{\max} < T$.

Literature Review. So far there is only one research paper that considers multicriteria scheduling with limited machine availability constraint [6]. In the paper, Huo and Zhao give optimal polynomial algorithms for three problems: (1) $P_{1,1} \mid r - a, prmt \mid \sum C_j / C_{\max}$; (2) $P_{1,1} \mid r - a, prmt \mid C_{\max} / \sum C_j$; and (3) $P_2 \mid r - a, prmt \mid C_{\max} / \sum C_j$ in which both machines are unavailable during an interval $[t, t + x)$ and at most one machine is unavailable at any other time.

In the following we will review the relevant results in the area of bi-criteria scheduling and in the area of scheduling with limited machine availability, respectively. We will survey the results concerning with makespan and total completion time only. For more details about multicriteria scheduling, see [2], [1], [14], [5] and the references therein. For details about scheduling with limited machine availability, see the surveys [10], [13] and [12].

Research on multicriteria scheduling problems on parallel machines has not been dealt with adequately in the literature. Gupta et al. ([4]) proposes an exponential algorithm to solve optimally the bi-criteria problem of minimizing the weighted sum of makespan and mean flow time on two identical parallel machines. When preemption is allowed, $P \mid prmt \mid C_{\max} / \sum C_j$ and $P \mid prmt \mid$

$\sum C_j/C_{\max}$ are polynomially solved by Leung and Young in [7], and Leung and Pinedo in [8], respectively.

With limited machine availability, when there are multiple machines, if preemption is not allowed, most problems are NP-hard. When preemption is allowed and the machines have limited availability constraint, the makespan problem is shown to be solvable in P by Liu and Sanlaville ([11]); additionally if the number of available machines does not go down by 2 within a period of $p_{\max}$ (which is the largest processing time of the jobs), Leung and Pinedo([9]) solved the total completion time minimization problem using PSPT (preemptive PSPT, i.e., at any time, when a machine becomes available for processing jobs, the job with the minimum remaining time gets scheduled.) rule.

When each machine has a release time, that is, machine M_i is only available in interval $[r_i, \infty)$, where $r_i \geq 0$, it is easy to show that SPT (shortest processing time first) rule minimizes $\sum C_j$ and one can modify the McNaughton rule (McNaughton (1959)) to minimizes the makespan.

New Contributions. In this paper, we study two problems: $P_m, r_i \mid r-a, prmt \mid \sum C_j/C_{\max} < T$ and $P_{m-1,1} \mid r-a, prmt \mid \sum C_j/C_{\max} < T$. We show both problems are in P by developing optimal algorithms. Note that a special case of our second problem is $m = 2$, which has been solved polynomially by the authors in [6]. While it is natural to conjecture the problem is still in P when $m > 2$, it turns out that the optimal algorithm itself and the proof of its optimality are much more complicated.

Organization. Our paper is organized as follows. In Section 2, we give some preliminary results. In Section 3, we study $P_m, r_i \mid r-a, prmt \mid \sum C_j/C_{\max} < T$. In Section 4, we study $P_{m-1,1} \mid r-a, prmt \mid \sum C_j/C_{\max} < T$. In Section 5, we draw the conclusion.

2 Preliminaries

As we mentioned before, SPT rule gives an optimal schedule for the total completion time when each machine has only a single unavailble interval starting from time 0. In general, however, SPT is not optimal when machines have arbitrary unavailable intervals. till, one can follow similar arguments from [6] and [9] to show that if there is at most one machine unavailable at any time, PSPT rule gives an optimal schedule.

Lemma 1. *For m machines, $m \geq 2$, such that at any time at most one machine is unavailable, PSPT generates an optimal schedule for $\sum C_j$.*

When the machines have arbitrary unavailable intervals, for the bi-criteria objective, $\sum C_j/C_{\max} \leq T$, one can show that the jobs completes in SPT order, see the following lemma.

Lemma 2. *For m machines with arbitrary unavailability constraints, $m \geq 2$, there is an optimal schedule for the objective $\sum C_j/C_{\max} \leq T$, such that if $p_i < p_j$, then $C_i \leq C_j$.*

Throughout this paper, we assume jobs J_1, J_2, ..., J_n are indexed in nondecreasing order of their processing times, i.e., $p_1 \leq p_2 \leq \ldots \leq p_n$.

3 $P_m, r_j \mid r - a, prmt \mid \sum C_j/C_{\max} \leq T$

In this section, we will develop an optimal algorithm for $P_m, r_j \mid prmt \mid \sum C_j/C_{\max} \leq T$. Here we consider $C^*_{max} \leq T \leq C'_{max}$ where C^*_{max} is the optimal makespan of problem $P_m, r_j \mid r - a, prmt \mid C_{\max}$ which can be solved by Liu and Sanlaville ([11]) and C'_{max} is the makespan of the schedule produced by PSPT rule. Apparently, if $T < C^*_{max}$, no feasible schedule exists and if $T > C'_{max}$, we can simply apply PSPT rule to the problem.

Let S be an empty schedule initially. The basic idea of our algorithm is to iteratively add jobs J_j, $1 \leq j \leq n$, in SPT order to S , subject to the condition that all the remaining jobs can finish by T. We can show that in order to check whether the remaining jobs can finish by T, it is sufficient to check whether the last $m-1$ jobs (or all the remaining jobs J_{j+1}, $\cdots$, J_n if $n - j < m - 1$) can finish by T. The specific procedure is described below.

Let n' be the number of jobs we are going to schedule as early as possible subject to the condition that the last $m-1$ jobs can finish by T. Initially, $n' = n$, and n' may be updated during the procedure. For each job J_j, $(1 \leq j \leq n')$, when we add it to S, we try to schedule it on the machine with the earliest idle time. For convenience, we use f_i and a_i to denote the earliest idle time and the total idle time on machine M_i respectively. Initially, we have $f_i = r_i$ and $a_i = T - r_i$, and after jobs are scheduled onto M_i, f_i will be updated to be the completion time of the last job scheduled on machine M_i and a_i will be updated as $a_i = T - f_i$. Without loss of generality, we assume that the machines are numbered such that $f_1 \leq f_2 \cdots \leq f_m$. So job J_j will be scheduled on machine M_1.

In order to make sure that the last $m-1$ jobs (or all the remaining jobs if $n - j < m - 1$) can finish by T, we first check if J_n can complete on M_2 by T, then check if J_{n-1} can complete on M_3 by T, and so on, in this order. During this process, we decide if a job J_k, $n-m+2 \leq k \leq n$, needs to preempt J_j, and if so, at what time and how much. Specifically, for each job J_k, $n \geq k \geq n-m+2$ (or $n \geq k \geq j+1$ if $n - j < m - 1$), we first check if $p_k \leq a_i$, $i = n - k + 2$. If so, it means that machine M_i has enough idle time to schedule J_k, and we say that this machine has a surplus $\sigma_i = a_i - p_k$; otherwise, only part of the job with length a_i can be scheduled on machine M_i, we say that the job has a deficit $\delta_k = p_k - a_i$. When a job J_k has deficit, we check if its deficit can be distributed to the machines with surplus. We check the machines in the order of M_{i-1}, M_{i-2},$\cdots$, M_2. If a machine $M_l (i - 1 \geq l \geq 2)$ has surplus σ_l and the surplus is more than the deficit, i.e. $\sigma_l \geq \delta_k$, we update $\sigma_l = \sigma_l - \delta_k$, $\delta_k = 0$

and we continue to schedule the next job J_{k-1} on machine M_{i+1}; otherwise, we update $\delta_k = \delta_k - \sigma_l$ and $\sigma_l = 0$, and we continue to check the next machine with surplus. If after we consider all the machines $M_l (i-1 \geq l \geq 2)$, we still have $\delta_k > 0$, then this is the amount that job J_k has to be scheduled on M_1. We try to schedule this part of J_k on M_1 as late as possible. So we first backward schedule J_k to the idle interval starting from f_i if there is any. If there is still part of J_k not scheduled, we schedule the remaining part of the job J_k to those intervals where J_j is scheduled but J_k is not, reschedule the replaced part of job J_j to the idle time on M_1 as early as possible, and set $n' = k-1$.

We repeat this procedure until all the last $m-1$ jobs (or all the remaining jobs if $n - j < m - 1$) are checked. We update the processing times of these $m-1$ jobs by reducing the part scheduled before the completion time of job J_j, fix the schedule of J_j and the jobs scheduled before J_j on machine M_1, remove the jobs that are scheduled after J_j on M_1. If J_j finishes at time T, or $j = n'$, we stop the procedure and go to next step; otherwise, we schedule next job J_{j+1} in a similar way.

If the procedure is stopped, we must have either $C_j = T$ or all the jobs $J_j (1 \leq j \leq n')$ have been scheduled. If there are still jobs unscheduled, that is, $n' < n$, we schedule all the remaining jobs $J_n, \cdots, J_{n'+1}$ one by one, in this order, on machine $M_2, \cdots, M_{n-n'+1}$ respectively. We schedule job J_k, $n \geq k \geq n'+1$ backwards from time T on machine M_i, $i = n-k+2$. If it cannot be completely scheduled on machine M_i, we schedule the unfinished part of job J_k backward during the idle intervals on machines $M_{i-1}, M_{i-2}, \cdots, M_2$, in this order. In case there is an overlap of J_k on $M_l (i-1 \geq l \geq 2)$ with part of J_k on other machines, we swap the overlapped J_k on M_l with the job $J_{n-(l-2)}$ on M_l. The formal algorithm is presented in the following.

Algorithm1
Input: r_i for $i = 1, \cdots, m$;
p_j for $j = 1, \cdots, n$, and $p_1 \leq p_2 \leq \cdots \leq p_n$.
Output: a schedule S

1. Let S be an empty schedule.
2. Let $f_i = r_i$ for all machine M_i, $i = 1, \cdots, m$.
3. Let J_j be the next job in SPT order, initialize $j = 1$
4. $n' = n$
5. While $j \leq n'$
   (a) Renumber the machines so that $f_1 \leq f_2 \cdots \leq f_m$.
   (b) Let $a_i = T - f_i$ for $i = 1, \cdots, m$.
   (c) Schedule J_j on machine M_1, update f_1, a_1.
   (d) Let $k = n$ and $i = 2$
   (e) While $k \geq j+1$ and $i \leq m$
       if $a_i \geq p_k$
           $\sigma_i = a_i - p_k$, $\delta_k = 0$
       else
       - $\sigma_i = 0$, $\delta_k = p_k - a_i$

$l = i - 1$
while $\delta_k > 0$ and $l \geq 2$
if $\sigma_l > \delta_k$, then $\sigma_l = \sigma_l - \delta_k$ and $\delta_k = 0$
else $\delta_k = \delta_k - \sigma_l$ and $\sigma_l = 0$
$l = l - 1$
if $\delta_k > 0$ starting from f_i backward, schedule the remaining part of J_k to the idle time interval
machine M_1 and update δ_k
if we still have $\delta_k > 0$ continue backward schedule the remaining part of J_k to the intervals where J_j is scheduled, and reschedule this part of J_j to the idle time interval M_1 as early as possible.
Update f_1, a_1.
set $n' = k - 1$
$i = i + 1$, $k = k - 1$

(f) Update the processing times of jobs $J_n, \ldots, J_{n-m+2}$ (or J_{j+1} if $n - j < m-1$) by reducing the part of these jobs scheduled before the completion time of job J_j on machine M_1
(g) Remove the jobs scheduled after J_j on M_1
(h) If the completion time of job J_j is T, set $n' = j$
(i) $j = j + 1$

6. $k = n$ and $i = 2$
7. While $k > n$
backward schedule J_k on M_i from T as much as possible.
if $a_i \geq p_k$, then $\sigma_i = a_i - p_k$, $\delta_k = 0$
else $\sigma_i = 0$, $\delta_k = p_k - a_i$
$l = i - 1$
while $\delta_k > 0$ and $l \geq 2$
if $\sigma_l > 0$
backward schedule J_k on M_l starting from the last idle time of length
$\min(\sigma_l, \delta_k)$. If some part of J_k on M_l overlaps with part of J_k that is
already scheduled on some other machines, swap this part of J_k on M_l
with same length of the job on M_l, $J_{n-(l-2)}$, from those intervals where
J_k is not scheduled
if $\sigma_l > \delta_k$, then $\sigma_l = \sigma_l - \delta_k$ and $\delta_k = 0$
else $\delta_k = \delta_k - \sigma_l$ and $\sigma_l = 0$
$l = l - 1$
8. Return S

We use an example to explain Algorithm1. There are 4 machines with release time 0, 3, 3, 10, respectively. We need to schedule 5 jobs before time $T = 12$. The processing times of $J_1, \ldots, J_5$ are 5, 6, 7, 7, 7, respectively. After we schedule J_1 on M_1 during $[0, 5)$, we check if J_5, J_4, and J_3 can finish before 12 on M_2, M_3 and M_4, respectively. Since $p_5 = 7 < a_2 = 9$, we have $\delta_5 = 0$, and $\sigma_2 = 2$. Similarly, $\delta_4 = 0$, and $\sigma_3 = 2$. For J_3, initially $\delta_3 = 5$. We reduce the deficit by utilizing the surpluses σ_3, σ_2. Now, $\delta_3 = 1$ and we backward schedule it on M_1 during the interval $[9, 10)$. After this process, we know that given the current schedule of J_1, all the remaining jobs can finish by 12. So we fix the schedule before the completion time of J_1, and remove the schedule of the remaining jobs from the schedule, see Figure 1 (1). Next reorder the machines, we schedule J_2 on M_1 (the original M_2) during interval $[3, 9)$, and do the check in a similar way. This time, however, $\sigma_2 = 2$ and $\sigma_3 = 0$. For J_3, initially $\delta_3 = 5$. After utilizing σ_2 and the interval $[9, 10)$ on M_1, J_3 has to preempt J_2 during the interval $[7, 9)$ and so J_2 is delayed and finished at 12. By the algorithm, we update the processing time of job J_3 by reducing the part scheduled before J_2 and stop Step 5. Then we schedule J_5, J_4 and J_3 according to Step 7. See Figure 1 (2) for the final schedule.

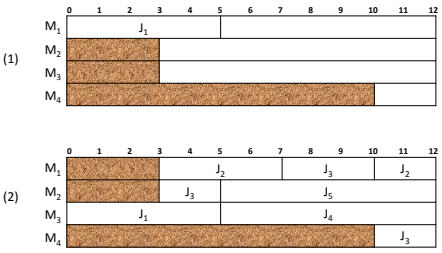


Fig. 1. Illustration of Algorithm 1

Lemma 3. *The order $p_n \geq p_{n-1} \geq \cdots \geq p_{n-m+2} \geq p_{n-m+1} \geq \cdots \geq p_1$ is always maintained even if p_k is updated for some $n \geq k \geq n - m + 2$ in some iteration in the algorithms.*

Proof. We show the Lemma is true after each time we schedule a job J_j. We first show that the relative order of the lengths of the last $(m-1)$ jobs is maintained. Let J_{k+1} and J_k be two of the last $(m-1)$ jobs. According to the algorithm, after J_j is scheduled on M_1, we check if job J_{k+1} can complete by T on machine M_{i-1} and if job J_k can complete by T on machine M_i, where $i = n - k + 2$.

If p_{k+1} is unchanged in Step 5, we must have $p_{k+1} \geq p_k$. Otherwise, part of J_{k+1} is scheduled before the completion time of job J_j on machine M_1. Let p'_{k+1} be the new length of job J_{k+1} after it is updated. We know $p'_{k+1} \geq a_{i-1}$. Furthermore, there is no surplus on any machine M_{i-1}, $\cdots$, M_2, and no idle time before f_{i-1}. If job J_k can be fully scheduled on machine M_i, we have $p_k \leq a_i \leq a_{i-1} \leq p'_{k+1}$. Otherwise, part of job J_k will be scheduled on machine M_1 backward from time f_i. Let p'_k be the new length of job J_k

after it is updated. Since there is no idle time before f_{i-1} on M_1, p'_k is at most $T - f_{i-1} = a_{i-1}$ at the end of Step 5. Thus $p'_k \leq a_{i-1} \leq p'_{k+1}$ and the order is maintained.

Now we show that $p_{n-m+1} < p_{n-m+2}$. If p_{n-m+2} is not updated in Step 5, then $p_{n-m+2} \geq p_{n-m+1} \geq \cdots \geq p_1$. Otherwise, let p'_{n-m+2} be the new length of job J_{n-m+2}. We must have $p'_{n-m+2} \geq a_m$ and there is no idle time before f_m on M_1. Furthermore, $\sigma_l = 0$, for all l, $2 \leq l \leq m$. In this case, all the remaining jobs have to be and can be scheduled on the first machine, which means we must have $a_m \geq a_1 \geq p_{n-m+1}$. Thus $p'_{n-m+2} \geq a_m \geq a_1 \geq p_{n-m+1}$. Since the processing time is unchanged for $p_1, p_2, \ldots, p_{n-m+1}$. The relative order of the (remaining) processing time is maintained.

We have the following two Lemmas about Algorithm1. Due to space limit, we omit the proofs.

Lemma 4. *Algorithm1 schedules the first n' jobs as soon as possible as long as the last $m-1$ jobs can finish before T and can be implemented in polynomial time.*

Lemma 5. *Algorithm1 finds a feasible schedule S of the n jobs.*

Theorem 1. *Algorithm1 returns an optimal schedule S for $P_m, r_j \mid r-a, prmt \mid \sum C_j/C_{\max} \leq T$ in polynomial time.*

Proof. We prove that S is optimal by showing there exists an optimal schedule that schedules the first n' jobs as early as possible subject to the last $m-1$ jobs can finish by T. Assume there is an optimal schedule S^* that does not schedule the jobs as early as possible. Let J_i be the first job in S^* that is not scheduled as early as possible and let t_1 be the first time that J_i is not scheduled in S^* but scheduled in S. That is, all the jobs with index less than i are scheduled in S^* exactly the same as in S and job J_i is scheduled in S^* exactly the same as S before time t_1. We will prove that we can always convert S^* such that job J_i can be scheduled at t_1 without increasing the total completion time of jobs in S^*.

Consider all the jobs $J_j (j > i)$ scheduled at t_1 in S^*. By Lemma 1, we have $C_j(S^*) \geq C_i(S^*)$. If there exists a time t' and a job J_j such that J_j is scheduled at t_1 but not at t' and J_i is scheduled at t', we can exchange J_j at t_1 with J_i at t'. The completion times of J_i and J_j are not increased and the completion times of other jobs are unchanged, so the total completion time is not increased.So in the following, we assume t' does not exist in S^*; i.e. in S^*, at any time after t_1, if J_i is scheduled, then J_j must also be scheduled.

For the convenience, we define $(J_{i_1}(t_{i_1}), J_{i_2}(t_{i_2}), \cdots, J_{i_p}(t_{i_p}))$ to be a " sequence" of a schedule, if we can reschedule J_{i_k}, from t_{i_k} to $t_{i_{k+1}}$ for $1 \leq k \leq p-1$ without increasing the total completion time of the schedule. And a sequence

$(J_{i_1}(t_{i_1}), J_{i_2}(t_{i_2}), \cdots, J_{i_p}(t_{i_p}))$ is an "exchange sequence" of a schedule if we can reschedule J_{i_k}, from t_{i_k} to $t_{i_{k+1}}$ (or t_{i_1} if $k = p$) without increasing the total completion time of the schedule. It is easy to see for this to be true, we must have J_{i_k}, $1 \leq k \leq p$, is scheduled at t_{i_k}, but not at $t_{i_{k+1}}$ (or t_{i_1} if $k = p$).

Depending on the completion time of the jobs scheduled at t_1 in S^*. we have the following two cases. We will show that in each case, there is an exchange sequence in S^* so that we can reschedule J_i at t_1 in S^*.

Case1: There exists a job J_j at t_1, $j > i$, $C_j \neq T$.
Note that neither J_i nor J_j is scheduled at C_j in S^*, and both J_i and J_j are scheduled at $C_i - 1$ in S^*. Since the number of available machines at C_j is not less than that at $C_i - 1$, there must exist a job J_l $(l > i)$ (or idle time/dummy job) which is scheduled at C_j but not scheduled at $C_i - 1$.
We can convert S^* based on the exchange sequence $(J_j(t_1), J_l(C_j), J_i(C_i - 1))$. In this way, C_i is decreased by at least 1, C_j is increased by 1 and C_l is not increased, all other jobs have same completion time as before, and the total completion time is not increased.

Case2: For all jobs J_j at t_1 such that $j > i$, we have $C_j = T$.
Since J_i is scheduled at t_1 in S but not in S^* and the number of available machines at t_1 is fixed, there must exists a job J_j scheduled at t_1 in S^* but not in S at time t_1. By induction, all the jobs J_j, $j < i$, have exactly the same schedule in S and S^*. So we must have $j > i$. Thus there must exist a time t_2 such that J_j is not scheduled at t_2 in S^* but is scheduled at t_2 in S. In return, there must exist a job J_k, which is scheduled at t_2 in S^* but not in S. Note that J_k cannot be a dummy job representing a machine idle at t_2. Otherwise, we can follow the sequence $(J_j(t_1), J_k(t_2))$ to move job J_j out of t_1 and move J_i to t_1. The obtained schedule has smaller total completion time which is a contradiction. Also, k cannot be smaller than i because by induction hypothesis, the schedule of these jobs are exactly the same in S and S^*. So we must have $k \geq i$. We can repeat the procedure and stop when we find a job J_p such that $C_p < T$. Since the number of jobs and the makespan of the schedule is finite, we must be able to stop at some point. In this way, we get a sequence $(J_j(t_1), J_k(t_2), \cdots, J_p(t_p))$. If $p = i$, then we can follow the exchange sequence $(J_j(t_1), J_k(t_2), \cdots, J_i(t_i))$ to convert S^* so that J_i is scheduled at t_1 in S^*. Otherwise, if there exists a time t_i that J_i is scheduled but J_p is not scheduled, then we can have the exchange sequence $(J_j(t_1), J_k(t_2), \cdots, J_p(t_p), J_i(t_i))$ to convert S^*. If there does not exist such time t_i, that is, at time $C_i - 1$, both J_i and J_p are scheduled, since at time C_p, both J_i and J_p are not scheduled, there must exist a job $J_l(l > i)$ which is scheduled at C_p but not at $C_i - 1$. We can follow the exchange sequence $(J_j(t_1), J_k(t_2), \cdots, J_p(t_p), J_l(C_p), J_i(C_i - 1))$

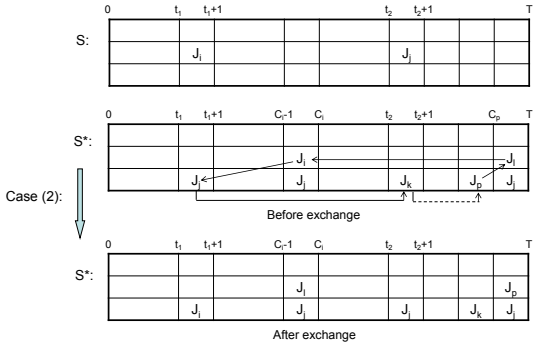


Fig. 2. Illustration of Theorem 1: Case2

to convert S^* so that J_i is scheduled at t_1 in S^*, see Figure 2. In summary, we can always reschedule S^* so that J_i is scheduled at t_1 without increasing the total completion time.

By repeating the above process, we can always convert S^* so that the schedule of the first n' jobs in S^* is exactly the same as that in S.

Note that the last $n - n'$ jobs complete exactly at T in S. And we can show that given the schedule of the first n' jobs exactly the same as in S, the remaining jobs have to complete at T in any schedule. This means that S must be optimal.

4 $P_{m-1,1} \mid r - a, prmt \mid \sum C_j / C_{\max} \leq T$

In this section, we study the problem $P_{m-1,1} \mid r - a, prmt \mid \sum C_j / C_{\max} \leq T$, where $C^*_{\max} \leq T < C'_{\max}$. We give an optimal polynomial time algorithm for this problem based on Algorithm1.

We use f_i to denote the completion time of the last job scheduled on machine M_i at the beginning of each iteration in our algorithm, and we assume that the machines are numbered so that $f_1 \leq f_2 \ldots \leq f_m$. Since preemption is allowed, for any unavailable interval between f_i and $f_{i+1} (1 \leq i \leq m-1)$, we can always assume that this unavailable period is on machine M_i; and for any unavailable interval after f_m, we can always assume that this interval is on machine M_m. It is easy to see that, at any time instant, if machine M_{i+1} is available, then M_i is also available. This assumption is just for ease of algorithm description and our proof. If job J_j is scheduled at certain time on a machine which is actually unavailable in S by our algorithm, we can always find a machine to schedule the job at the same time without rearranging the real unavailable intervals since preemptive is allowed.

Algorithm2

1. Let S be an empty schedule.
2. Let $f_i = 0$ for all machine M_i, $i = 1, \ldots, m$.
3. $n' = n$
4. Let J_j be the next job in SPT order, initialize $j = 1$
5. While $j \leq n'$
   (a) Renumber the machines that still have idle time so that $f_1 \leq f_2 \cdots \leq f_m$.
   (b) Rearrange the unavailable intervals, so that all the unavailable intervals between f_i and f_{i+1} are on machine M_i and all the unavailable intervals after f_m are on machine M_m.
   (c) Update a_i to be the total available time between f_i and T on machine M_i, $i = 1, \ldots, m$.
   (d) Schedule J_j on machine M_1,update f_1, a_1.
   (e) Let $k = n$ and $i = 2$
   (f) While $k \geq j + 1$ and $i \leq m$
       if $i = m$ rearrange the unavailable intervals $[s, t]$ where $s \geq f_1$, from M_m to M_1 update a_1, and a_m.
       if $a_i \geq p_k$
           $\sigma_i = a_i - p_k$, $\delta_k = 0$
       else
       - $\sigma_i = 0$, $\delta_k = p_k - a_i$,
         $l = i - 1$
         while $\delta_k > 0$ and $l \geq 2$
             if $\sigma_l > \delta_k$, $\sigma_l = \sigma_l - \delta_k$, $\delta_k = 0$
             else $\delta_k = \delta_k - \sigma_l$, $\sigma_l = 0$
             $l = l - 1$
         if $\delta_k > 0$ starting from T backward, schedue the remaining part of J_k to the idle itervals $[s, t)$ (if there is any) on M_1 such that either (1) $s > f_i$ and $[s, t)$ is an unavailable interval on machine M_i or (2) $t \leq f_i$.
             update δ_k
         if $\delta_k > 0$ continue to backward schedule the remaining part of J_k starting from f_i to the intervals where J_j is scheduled, and reschedule this part of J_j to the idle time interval M_1 as early as possible
             update f_1, a_1
             n' = k-1
       $i = i + 1$, $k = k - 1$
   (g) Update the processing times of jobs $J_n, \ldots, J_{n-m+2}$ (or J_{j+1} if $n - j < m-1$) by reducing the part of these jobs scheduled before the completion time of job J_j on machine M_1
   (h) Remove the jobs scheduled after J_j on M_1
   (i) If the completion time of job J_j is T set $n' = j$
   (j) $j = j + 1$

6. $k = n$ and $i = 2$
7. While $k \geq j$
   backward schedule J_k on M_i from T as much as possible.
   if $a_i \geq p_k$, then $\sigma_i = a_i - p_k$, $\delta_k = 0$
   else $\sigma_i = 0$, $\delta_k = p_k - a_i$
   $l = i - 1$
   while $\delta_k > 0$ and $l \geq 2$
      if $\sigma_l > 0$ backward schedule J_k on M_l starting from the last idle time of length $\min(\sigma_l, \delta_k)$. If some part of J_k on M_l overlaps with part of J_k that is already scheduled on some other machines, swap this part of J_k on M_l with same length of the job on M_l, $J_{n-(l-2)}$, from those intervals where J_k is not scheduled
         if $\sigma_l > \delta_k$, then $\sigma_l = \sigma_l - \delta_k$ and $\delta_k = 0$
         else $\delta_k = \delta_k - \sigma_l$ and $\sigma_l = 0$
      $l = l - 1$
8. Return S

We use an example to explain Algorithm2. There are 4 machines such that M_1 is unavailable during $[0, 3)$, M_3 is unavailable during $[4, 7)$, M_4 is unavailable during $[8, 12)$. See Figure 3 (0). We need to schedule 5 jobs before time $T = 12$. The processing times of $J_1, \ldots, J_5$ are 5, 6, 7, 10, 10, respectively. We first rearrange the unavailable intervals and renumber the machines so that all unavailable intervals are on the last machine M_4. Then we schedule J_1 on M_1 during $[0, 5)$, we check if J_5, J_4, and J_3 can finish before 12. Since $p_5 = 10 < a_2 = 12$, we have $\sigma_2 = 2$, and $\delta_5 = 0$. Similarly, $\sigma_3 = 2$, and $\delta_4 = 0$, see Figure 3 (1). For J_3, we first move the unavailable interval after time 5 to M_1. Then we get $\sigma_4 = 1$ $\delta_3 = 0$. So we fix the schedule of J_1, reorder the machines, see Figure 4 (3). Then we schedule J_2 on the new M_1 during interval $[0, 6)$, and do the check in a similar way. This time, however, J_4 has to utilize the surplus on M_2 (see Figure 4 (4)), and J_3 has to preempt J_2 and so J_2 is delayed. By the algorithm,

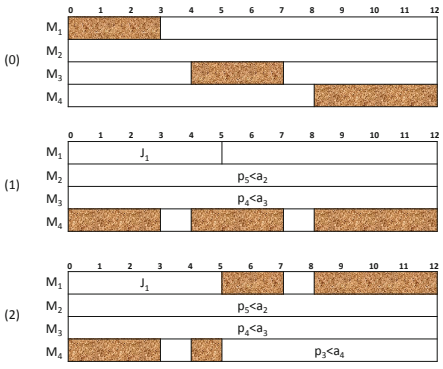


Fig. 3. Illustration of Algorithm 2

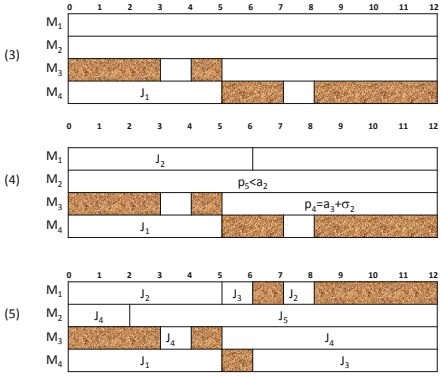


Fig. 4. Illustration of Algorithm 2

we update the processing time of J_3 and set $n' = 2$ and stop Step (e). Then, we schedule job 5, 4, 3 in this order based on Step 7. The final schedule is given Figure 4 (5).

Use similar argument, we can show the following.

Theorem 2. *Algorithm2 returns an optimal schedule S for* $P_{m-1,1} \mid r-a, prmt \mid \sum C_j/C_{\max} \leq T$.

5 Conclusion

In this paper, we study two bi-criteria scheduling problems subject to the machine unavailability constraint. Our first problem, $P_m, r_i \mid r - a, prmt \mid \sum C_j/C_{\max} \leq T$, concerns the case that each machine has a release time after which the machine is always available. We show that an optimal schedule can be obtained in polynomial time. The second problem, $P_{m-1,1} \mid r - a, prmt \mid \sum C_j/C_{\max} \leq T$, assumes that each machine can have multiple unavailable intervals, but at any time there is at most one machine unavailable. We also design an optimal algorithm for this problem by modifying the first algorithm. Both algorithms and the proofs are quite involved and subtle.

References

1. Chen, C.L., Bulfin, R.L.: Complexity of single machine, multicriteria scheduling problems. European Journal of Operational Research 70, 115–125 (1993)
2. Dileepan, P., Sen, T.: Bicriteria static scheduling research for a single machine. OMEGA 16, 53–59 (1988)
3. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling, a survey. Annals of Discrete Mathematics 5, 287–326 (1979)

4. Gupta, J.N.D., Ho, J.C., Webster, S.: Bicriteria optimisation of the makespan and mean flowtime on two identical parallel machines. Journal of Operational Research Society 51(11), 1330–1339 (2000)
5. Hoogeveen, J.A.: Multicriteria scheduling. European Journal of Operational Research 167(3), 592–623 (2005)
6. Huo, Y., Zhao, H.: Bicriteria Scheduling Concerned with Makespan and Total Completion Time Subject to Machine Availability Constraints. Theoretical Computer Science 412, 1081–1091 (2011)
7. Leung, J.Y.-T., Young, G.H.: Minimizing schedule length subject to minimum flow time. SIAM Journal on Computing 18(2), 314–326 (1989)
8. Leung, J.Y.-T., Pinedo, M.L.: Minimizing total completion time on parallel machines with deadline constraints. SIAM Journal on Computing 32, 1370–1388 (2003)
9. Leung, J.Y.-T., Pinedo, M.L.: A Note on the scheduling of parallel machines subject to breakdown and repair. Naval Research Logistics 51, 60–72 (2004)
10. Ma, Y., Chu, C., Zuo, C.: A survey of scheduling with deterministic machine availability constraints. Computers & Industrial Engineering 58(2), 199–211 (2010)
11. Liu, Z., Sanlaville, E.: Preemptive scheduling with variable profile, precedence constraints and due dates. Discrete Applied Mathematics 58, 253–280 (1995)
12. Saidy, H., Taghvi-Fard, M.: Study of Scheduling Problems with Machine Availability Constraint. Journal of Industrial and Systems Engineering 1(4), 360–383 (2008)
13. Schmidt, G.: Scheduling with limited machine availability. European Journal of Operational Research 121(1), 1–15 (2000)
14. T'kindt, V., Billaut, J.C.: Multicriteria Scheduling: Theory, Models and Algorithms. Springer, Heidelberg (2002)

Zero-Sum Flow Numbers of Hexagonal Grids

Tao-Ming Wang* and Guang-Hui Zhang

Department of Applied Mathematics
Tunghai University
Taichung, Taiwan 40704

Abstract. As an analogous concept of nowhere-zero flows for directed and bi-directed graphs, we consider zero-sum flows for undirected graphs in this article. For an undirected graph G, a **zero-sum k-flow** is an assignment of non-zero integers whose absolute values less than k to the edges, such that the sum of the values of all edges incident with each vertex is zero. Furthermore we generalize the notion via considering a combinatorial optimization problem, which is to calculate the **zero-sum minimum flow number** of a graph G, namely, the least integer k for which G may admit a zero-sum k-flow. The **Zero-Sum 6-Flow Conjecture** was raised by Akbari et al. in 2009: If a graph with a zero-sum flow, it admits a zero-sum 6-flow. It turns out that this conjecture was proved to be equivalent to the classical Bouchet 6-flow conjecture for bi-directed flows. In this paper, we study zero-sum minimum flow numbers of graphs induced from plane tiling by regular hexagons in an arbitrary way, namely, the **hexagonal grid graphs**. In particular we are able to verify the Zero-Sum 6-Flow Conjecture for the class of hexagonal grid graphs by determining the zero-sum flow number of any non-trivial hexagonal grid graph is 3 or 4. We further use the concept of dual graphs to specify classes of infinite families of hexagonal grid graphs with minimum flow numbers 3 and 4 respectively. Further open problems are included.

Keywords: zero-sum flow, zero-sum minimum flow number, hexagonal grid, square grid, triangular grid.

1 Background and Motivation

Throughout this paper, all terminologies and notations on graph theory can be referred to the textbook by D. West[10]. Let G be a directed graph. A **nowhere-zero flow** on G is an assignment of non-zero integers to each edge such that for every vertex the Kirchhoff current law holds, that is, the sum of the values of incoming edges is equal to the sum of the values of outgoing edges. A **nowhere-zero k-flow** is a nowhere-zero flow using edge labels with maximum absolute value $k-1$. Note that for a directed graph, admitting nowhere-zero flows is

* The corresponding author with e-mail address wang@go.thu.edu.tw and the research is partially supported by the National Science Council of Taiwan under project NSC-101-2115-M-029-001.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 339–349, 2013.

independent of the choice of the orientation, therefore one may consider such concept over the underlying undirected graph. A celebrated conjecture of Tutte in 1954 says that every bridgeless graph has a nowhere-zero 5-flow. F. Jaeger showed in 1979 that every bridgeless graph has a nowhere-zero 8-flow[5], and P. Seymour proved that every bridgeless graph has a nowhere-zero-6-flow[6] in 1981. However the original Tutte's conjecture remains open. There is a more general concept of a nowhere-zero flow that uses bidirected edges instead of directed ones, first systematically developed by Bouchet[4] in 1983. Bouchet raised the conjecture that every bidirected graph with a nowhere-zero integer flow has a nowhere-zero 6-flow, which is still unsettled.

Recently another analogous nowhere-zero flow concept has been studied, as a special case of bi-directed one, over the undirected graphs by S. Akbari et al.[1,2] in 2009 and 2010, which is defined as follows:

Definition 1. *For an undirected graph G, a* **zero-sum flow** *is an assignment of non-zero integers to the edges such that the sum of the values of all edges incident with each vertex is zero. A* **zero-sum k-flow** *is a zero-sum flow whose values are integers with absolute value less than k.*

S. Akbari et al. raised a conjecture (called **Zero-Sum 6-Flow Conjecture**) for zero-sum flows similar to the Tutte's 5-flow Conjecture for nowhere-zero flows as follows: If G is a graph with a zero-sum flow, then G admits a zero-sum 6-flow. It was proved in 2010 by Akbari et al. [1] that the above Zero-Sum 6-Flow Conjecture is equivalent to the Bouchet's 6-Flow Conjecture for bidirected graphs, and the existence of zero-sum 7-flows for regular graphs were also obtained. Based upon the results, they raised another weaker conjecture for regular graphs: If G is a r-regular graph with $r \geq 3$, then G admits a zero-sum 5-flow.

In literature a more general concept **minimum flow number**, which is defined as the least integer k for which a graph may admit a k-flow, has been studied for both directed graphs and bidirected graphs. We extend the concept in 2011 to the undirected graphs and call it the zero-sum minimum flow number [9]:

Definition 2. *Let G be a undirected graph. The* **zero-sum minimum flow number** *$F(G)$ is defined as the least number of k for which G may admit a zero-sum k-flow. $F(G) = \infty$ if no such k exists.*

In particular we obtain a characterization of graphs with flow number 2, and also a characterization of 3-regular graphs with flow number 3 among other results[8]. Note that the related result were presented in the FAW 2012 conference by the first author in Beijing. We introduce the basic properties and previous results of the zero-sum minimum flow numbers in later section. On the other hand, it is well known that grids are extremely useful in all areas of computer science. One of the main usage, for example, is as the discrete approximation to a continuous domain or surface. Numerous algorithms in computer graphics, numerical analysis, computational geometry, robotics and other fields are based on grid computations.

It is known that there are only three possible types of regular tessellations, which are tilings made up of squares, equilateral triangles, and hexagons. We consider and study the minimum flow numbers of graphs induced from plane tiling by regular polygons in an arbitrary way. Formally, a square grid, or a **square grid graph** is induced by an arbitrary finite subset of the infinite integer lattice grid $\mathbb{Z} \times \mathbb{Z}$. The vertices of a square grid are the lattice points, and the edges connect the points which are at unit distance from each other. The infinite grid $\mathbb{Z} \times \mathbb{Z}$ may be viewed as the set of vertices of a regular tiling of the plane with unit squares. Another type is with equilateral triangles, which defines an infinite triangular grid in a similar way. A **triangular grid graph** is a graph induced by an arbitrary finite subset of the infinite triangular grid. One more type of plane tiling is with regular hexagons which defines an infinite hexagonal grid, and the graph induced by an arbitrary finite subset of the infinite hexagonal grid is called a **hexagonal grid graph**. (See Figure 1) A hexagonal grid graph is also named a **honeycomb graph** in literature. We pay attention to hexagonal grid graphs in this article.

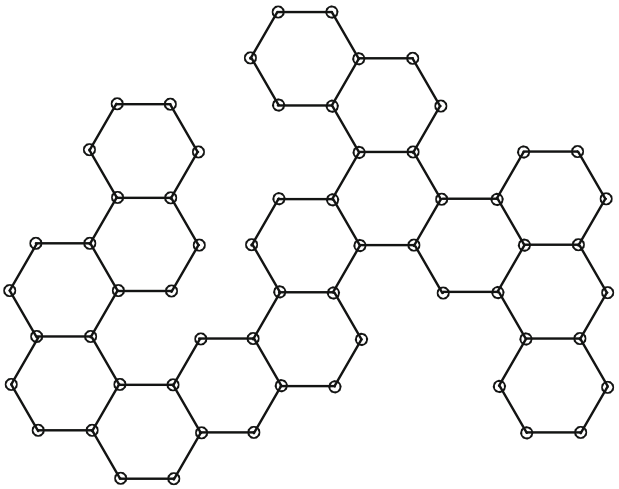


Fig. 1. Example of a Hexagonal Grid Graph

Note that Akbari. et al. showed that in [2] if **Zero-Sum 6-Flow Conjecture** is true for $(2, 3)$-graphs (in which every vertex is of degree 2 or 3), then it is true for any graph. Henceforth the study can be reduced to $(2, 3)$-graphs. It is clear non-trivial hexagonal grid graphs are a special class of $(2, 3)$-graphs. Therefore in this paper we focus the study over the zero-sum flow numbers for hexagonal grid graphs. In particular we are able to verify the Zero-Sum 6-Flow Conjecture for the class of hexagonal grid graphs by determining the zero-sum flow number of any non-trivial hexagonal grid graph is 3 or 4. We further use the concept of dual graphs to specify classes of infinite families of hexagonal grid graphs with minimum flow numbers 3 and 4 respectively.

2 Preliminaries of Zero-Sum Minimum Flow Numbers

In 2011 [9] we generalize the notion zero-sum flows by considering a combinatorial optimization problem, which is to find the **zero-sum minimum flow number** of a graph G, namely the least number of k for which G may admit a zero-sum k-flow. Obviously the zero-sum minimum flow numbers provide with more detailed information regarding zero-sum flows. For example, we may restate the previously mentioned Zero-Sum Conjecture as follow: Suppose a undirected graph G has a zero-sum flow, then $F(G) \leq 6$. We showed in [8] some general properties of small minimum flow numbers, so that the calculation of zero-sum minimum flow numbers becomes easier and efficient. In particular we obtained the following pretty useful technical lemma for the characterization of graphs with minimum flow number 2 which is used frequently in this paper, and we provide with a proof for completeness here:

Lemma 1. (T.-M. Wang and S.-W. Hu, [8]) *A graph G has zero-sum minimum flow number $F(G) = 2$ if and only if G is Eulerian with even size (even number of edges) in each component.*

Proof. Without loss of generality, we may assume G is connected. We start showing the necessary part. Since a graph G has flow index $F(G) = 2$ meaning it admits a zero-sum 2-flow, thus the edge function $f(e) \in \{1, -1\}$. For each vertex $v \in V(G)$, the number of incident edges labeled 1 must equal to the number of incident edges labeled -1. Note that both numbers are equal to $\frac{1}{2}deg(v)$, therefore $deg(v)$ must be even, and G is Eulerian. On the other hand, the number of all 1-edges (or (-1)-edges) in G is $\frac{1}{2}\sum_{v\in V(G)}(\frac{1}{2}deg(v)) = \frac{1}{2}|E(G)|$ which is an integer, so $|E(G)|$ are even. Conversely, to show the sufficiency we label the edges in an Euler tour of G by 1 and -1 alternatively. Then every vertex is incident with the same number of 1-edges and (-1)-edges, including the starting(ending) vertex, since the number of edges is even. Therefore it is a zero-sum 2-flow in G. ■

In [8] we also calculate the zero-sum flow numbers of regular graphs, which is closely related to the zero-sum 5-flow conjecture for regular graphs. Recently it is known that the zero-sum 5-flow conjecture for regular graphs was nearly completely resolved by S. Akbari and other authors [3], except the case for 5-regular graphs. We study the zero-sum flows more recently and obtain certain results toward to these conjectures. Among other results we show that in [8] that every bridgeless 5-regular graph G admits a 5-flow, which strengthens the zero-sum 5-flow conjecture for regular graphs.

In next section we calculate the zero-sum minimum flow numbers for various types of graphs induced from the plane tiling by hexagons.

3 Zero-Sum Flow Numbers of Hexagonal Grid Graphs

It is well known for the notion of the dual graph $D(G)$ of a plane graph G for a fixed plane drawing representation of G embedded in a sphere or the plane.

Note that generally the dual graph of a hexagonal grid is (a partial subgraph of) a triangular grid. See for example Figure 2.

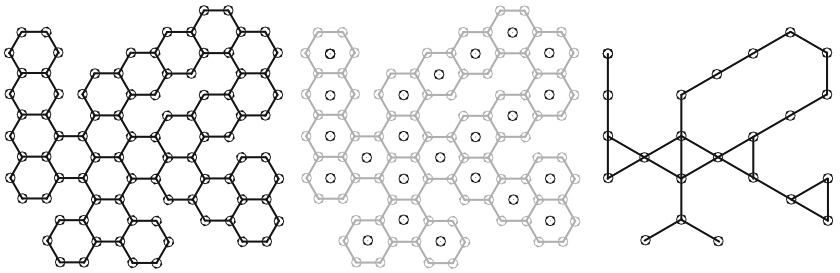


Fig. 2. A Hexagonal Grid Graph with its Dual Graph

Now we set up fundamental symbols for the trivial regular hexagon labeled ± 1 and ± 2 over the edges as in Figure 3. The symbol $\mathcal{I}$ stands for the trivial regular hexagon edge-labeled 1 and -1 consecutively with zero-sums. $-\mathcal{I}$ and $\pm 2\mathcal{I}$ stand for the ones with zero-sums using labels of $\mathcal{I}$ multiplied by -1 and ± 2 respectively. Note that in figures below, the weight of the overlapping edge for any two neighboring fundamental symbols are summed up from both patterns.

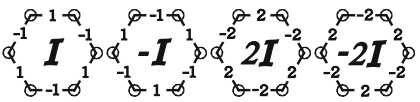


Fig. 3. Fundamental Hexagons with Zero-Sum 2-Flows and 3-Flows

Note that also the zero-sum minimum flow number of the trivial regular hexagon is 2. The following gives the optimal upper bound for the minimum flow number of any finite non-trivial hexagonal grid graph:

Theorem 1. *The infinite hexagonal grid graph $\widetilde{H}$ admits a zero-sum 3-flow and $F(\widetilde{H}) = 3$. Moreover let H be any finite non-trivial hexagonal grid graph. Then $F(H) = 3$ or 4.*

Proof. Note that one obtains a zero-sum flow of the whole figure while patching together sub-figures with zero-sums in an arbitrary way of union. See Figure 4 and note that the weight of the overlapping edge for any two neighboring fundamental symbols are summed up from both patterns. Therefore one has a zero-sum 3-flow for the infinite hexagonal grid graph $\widetilde{H}$ using the fundamental figures $\pm\mathcal{I}$. On the other hand, it is impossible for $\widetilde{H}$ to admit a 2-flow due to the existence of odd degree vertices. Thus $F(\widetilde{H}) = 3$.

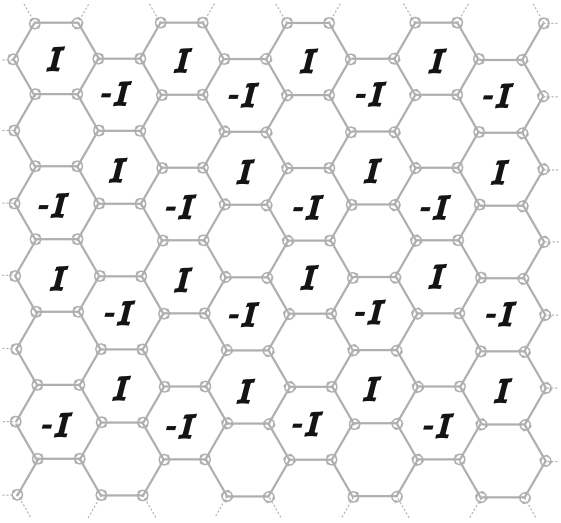


Fig. 4. A 3-Flow of the Infinite Hexagonal Grid $\widetilde{H}$

As for any finite non-trivial hexagonal grid graph, we obtain the bounds for the flow numbers via the labeling of the infinite hexagonal grid. It is not hard to check as in Figure 5 one has a zero-sum 4-flow for the infinite hexagonal grid graph $\widetilde{H}$, using the fundamental figures $\pm\mathcal{I}$ and $\pm 2\mathcal{I}$. Note that again the weight of the overlapping edge for any two neighboring fundamental symbols are summed up from both patterns.

Note that then any finite non-trivial hexagonal grid graph H may be treated a piece of finite sub-figure cut from the infinite hexagonal grid $\widetilde{H}$. Therefore, H admits a zero-sum 4-flow using exactly the same edge labels induced from those of $\widetilde{H}$ (see Figure 6). Thus by Lemma 1 the minimum flow numbers are 3 or 4 except that the trivial regular hexagon has flow number 2 as indicated in $\mathcal{I}$. ■

We also determine various classes of infinite families of hexagonal grid graphs with flow numbers 3 and 4 respectively. First we start with classes of flow numbers 3:

Theorem 2. *Let G be a non-trivial hexagonal grid graph with the dual graph $D(G)$ to be bipartite. Then $F(G) = 3$.*

Proof.

Note that if the dual graph $D(G)$ is bipartite, it is 2-colorable. Then using $\pm\mathcal{I}$ as two colors to put over the vertices of the dual graph. We see G admits a 3-flow with edge labeling this way and again by Lemma 1 the zero-sum flow number is 3. ■

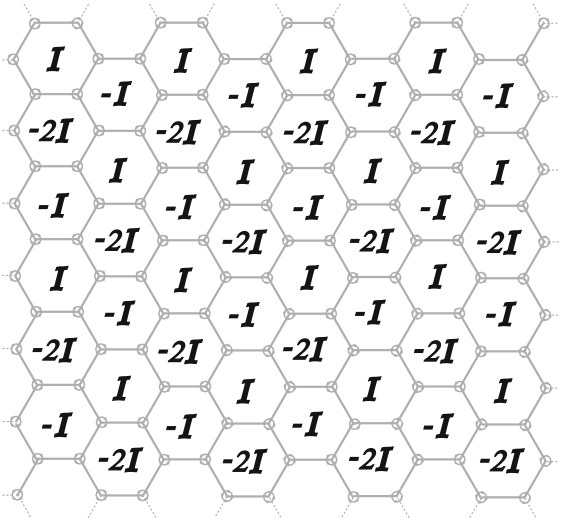


Fig. 5. A 4-Flow of the Infinite Hexagonal Grid $\widetilde{H}$

Therefore we may easily have the following examples of flow numbers 3 since their dual graphs are trees, thus bipartite:

Theorem 3. *Let G be a non-trivial hexagonal grid graph with the dual graph $D(G)$ consisting of multiple W_6 copies, for which one W_6 shares at most one edge with another copy of W_6. (see Figure 8) Then $F(G) = 3$.*

Proof
Note that if the dual graph $D(G)$ consists of W_6 copies for which one sharing at most one edge with another, one may fix it into a hexagonal grid graph by dropping the central vertex of each copy of W_6 (see Figure 8 for an example to reduce the dual graph). It is clear that the resulting reduced dual graph is bipartite. Hence by Theorem 2 we see $F(G) = 3$. ■

The following are examples of classes of infinitely many hexagonal grids with flow number 4:

Theorem 4. *Let G be a hexagonal grid graph with the dual graph $D(G)$ which contains a triangle with one degree 2 vertex (see Figure 9). Then $F(G) = 4$.*

Proof
Assume G admits a zero-sum 3-flow, which allows only labels $\pm 1, \pm 2$. See Figure 10 without loss of generality may assume $a = 1$ or 2. In both cases through detailed calculation one will reach contradictions for c and d, for either $c + d = 0$ or $c + d = \pm 3$. Therefore $F(G) = 4$. ■

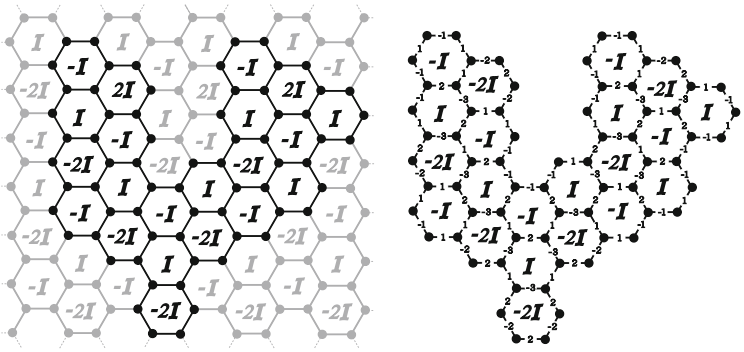


Fig. 6. A 4-Flow of Arbitrary Finite Hexagonal Grid Induced from $\widetilde{H}$

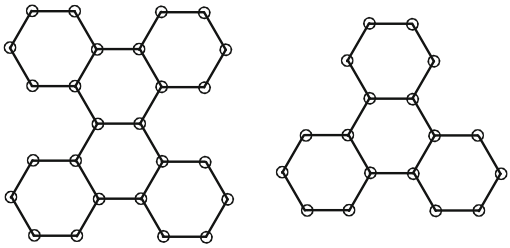


Fig. 7. Examples of Hexagonal Grids with Flow Number 3

Theorem 5. *Let G be a hexagonal grid graph with the dual graph $D(G)$. Suppose that $D(G)$ contains a kite with one degree 4 vertex or an antenna triangle with one degree 3 vertex (as in the Figure 11). Then $F(G) = 4$.*

Proof. Assume G admits a zero-sum 3-flow. The common figure of a hexagonal grid graph containing a kite or an antenna triangle as its dual graph can be seen in the Figure 12. Then without loss of generality we may assume $a = 1$ or 2. In both cases through detailed calculation one will reach contradiction for g and h, for either $g + h = 0$ or $g + h = \pm 3$. Therefore $F(G) = 4$. ■

As corollary one may determine the flow numbers of regular hexagonal cluster grids H_n, which are the graphs in Figure 13. Note that H_n contains diagrams in Figure 12 for each $n \geq 3$. Thus by the Theorem 5 we have:

Corollary 1. *The minimum flow number of the regular hexagonal cluster grid H_n of n layers are as follows:*

$$F(H_n) = \begin{cases} 2, & n = 1. \\ 3, & n = 2. \\ 4, & n \geq 3. \end{cases}$$

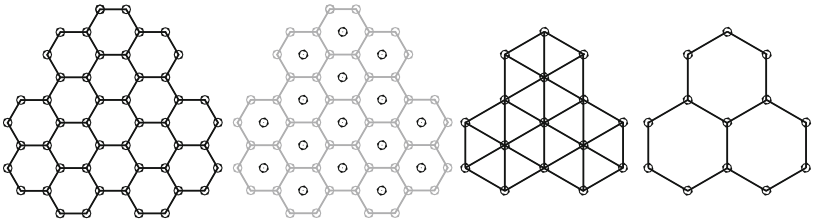


Fig. 8. Example of Hexagonal Grid with Dual Graph Multiple W_6 Copies

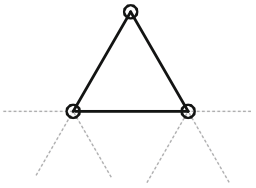


Fig. 9. Dual graph $D(G)$ contains a triangle with one degree 2 vertex

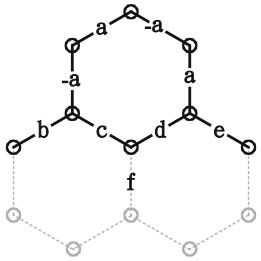


Fig. 10. Example of G whose dual contains a triangle with one degree 2 vertex

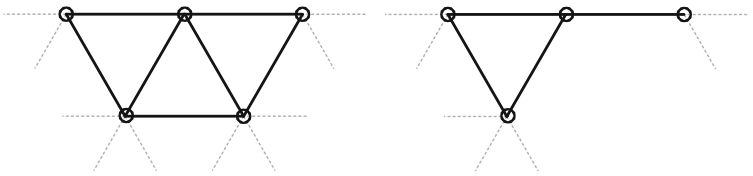


Fig. 11. Dual graph $D(G)$ contains a Kite or an Antenna Triangle

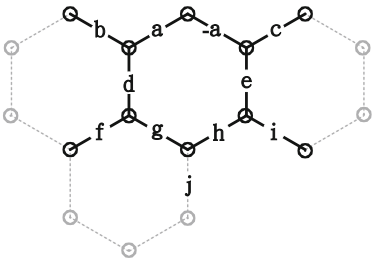


Fig. 12. Hexagonal Grid with Kite or Antenna Triangle as its Dual Graph

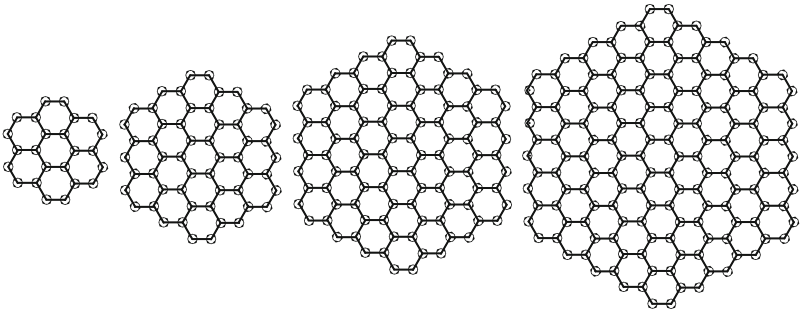


Fig. 13. Hexagonal Cluster H_2, H_3, H_4, H_5

4 Concluding Remark and Open Problem

In this paper we are able to determine that the zero-sum flow number of any non-trivial hexagonal grid graph is 3 or 4. We further find classes of infinite families of hexagonal grid graphs with minimum flow numbers 3 and 4 respectively. We also calculate as corollaries the zero-sum minimum flow numbers of infinite families of regular hexagonal grids.

However while one may calculate the zero-sum flow numbers of above classes of hexagonal grids, it is interesting to characterize completely the classes of non-trivial hexagonal graphs with zero-sum flow numbers 3 and 4 respectively. The zero-sum flow numbers of square grid graphs are not hard to calculate, while the complete characterizations of triangular grid graphs with various flow numbers and their optimal bounds are relatively nice open problems worth to work on.

Acknowledgement. The authors wish to express their sincere thanks to the referees for their detailed comments and suggestions.

References

1. Akbari, S., Daemi, A., Hatami, O., Javanmard, A., Mehrabian, A.: Zero-Sum Flows in Regular Graphs. Graphs and Combinatorics 26, 603–615 (2010)
2. Akbari, S., Ghareghani, N., Khosrovshahi, G.B., Mahmoody, A.: On zero-sum 6-flows of graphs. Linear Algebra Appl. 430, 3047–3052 (2009)
3. Akbari, S., Ghareghani, N., Khosrovshahi, G.B., Zare, S.: A note on zero-sum 5-flows in regular graphs. Electronic Journal of Combinatorics 19(2) (2012) # P7
4. Bouchet, A.: Nowhere-zero integral flows on a bidirected graph. J. Combin. Theory Ser. B 34, 279–292 (1983)
5. Jaeger, F.: Flows and generalized coloring theorems in graphs. J. Combin. Theory Ser. B 26(2), 205–216 (1979)
6. Seymour, P.D.: Nowhere-zero 6-flows. J. Combin. Theory Ser. B 30(2), 130–135 (1981)
7. Wang, T.-M., Hu, S.-W.: Constant sum flows in regular graphs. In: Atallah, M., Li, X.-Y., Zhu, B. (eds.) FAW-AAIM 2011. LNCS, vol. 6681, pp. 168–175. Springer, Heidelberg (2011)
8. Wang, T.-M., Hu, S.-W.: Zero-Sum Flow Numbers of Regular Graphs. In: Snoeyink, J., Lu, P., Su, K., Wang, L. (eds.) FAW-AAIM 2012. LNCS, vol. 7285, pp. 269–278. Springer, Heidelberg (2012)
9. Wang, T.-M., Hu, S.-W.: Nowhere-zero constant-sum flows of graphs. Presented in the 2nd India-Taiwan Conference on Discrete Mathematics, Coimbatore, Tamil Nadu, India (September 2011) (manuscript)
10. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice Hall, Englewood Cliffs (2001)

Pattern-Guided k-Anonymity

Robert Bredereck*, André Nichterlein, and Rolf Niedermeier

Institut für Softwaretechnik und Theoretische Informatik,
TU Berlin, Berlin, Germany
{robert.bredereck,andre.nichterlein,rolf.niedermeier}@tu-berlin.de

Abstract. We suggest a user-oriented approach to combinatorial data anonymization. A data matrix is called k-anonymous if every row appears at least k times—the goal of the NP-hard k-Anonymity problem then is to make a given matrix k-anonymous by suppressing (blanking out) as few entries as possible. We describe an enhanced k-anonymization problem called Pattern-Guided k-Anonymity where the users can express the differing importance of various data features. We show that Pattern-Guided k-Anonymity remains NP-hard. We provide a fixed-parameter tractability result based on a data-driven parameterization and, based on this, develop an exact ILP-based solution method as well as a simple but very effective greedy heuristic. Experiments on several real-world datasets show that our heuristic easily matches up to the established "Mondrian" algorithm for k-Anonymity in terms of quality of the anonymization and outperforms it in terms of running time.

1 Introduction

Making a matrix *k-anonymous*, that is, each row has to occur at least k times, is a classic model for (combinatorial) data privacy [12, 21].[1] The idea behind is that each row of the matrix represents an individual and the k-fold appearance of the corresponding row avoids that the person or object behind can be identified. To reach this goal, clearly some information loss has to be accepted, that is, some entries of the matrix have to be suppressed (blanked out); in this way, information about certain attributes (represented by the columns of the matrix) is lost. Thus, the natural goal is to minimize this loss of information when transforming an arbitrary data matrix into a k-anonymous one. The corresponding optimization problem k-Anonymity is NP-hard (even in special cases) and hard to approximate [1, 2, 3, 7, 20]. Nevertheless, it played a significant role in many applications, mostly relying on heuristic approaches for making a matrix k-anonymous [6, 13, 21].

* Supported by the DFG, research project PAWS, NI 369/10.

[1] We omit considerations on the recently very popular model of "differential privacy" [8] which has a more statistical than a combinatorial flavor. It is well-known that there are certain weaknesses of the k-anonymity concept when the anonymized data is used multiple times [5, 12]. Here, we focus on k-anonymity which due to its simplicity and good interpretability continues to be of interest in current applications.

M. Fellows, X. Tan, and B. Zhu (Eds.): FAW-AAIM 2013, LNCS 7924, pp. 350–361, 2013.

It was observed that care has to be taken concerning the "usefulness" (also in terms of expressiveness) of the anonymized data [18, 22]. Indeed, depending on the application that has to work on the k-anonymized data, certain entry suppressions may "hurt" less than others. E.g., considering medical data records, the information about eye color may be less informative than information about the blood pressure. Hence, it would be useful for the later user of the anonymized data to specify information that may help doing the anonymization process in a more sophisticated way. Thus, in recent work [4] we proposed a "pattern-guided" approach to data anonymization, in a way allowing the user to specify which combinations of attributes are less harmful to suppress than others. More specifically, the approach allows "pattern vectors" which may be considered as blueprints for the structure of anonymized rows—each row has to be matched with exactly one of the pattern vectors. The correspondingly proposed optimization problem [4], however, has the clear weakness that each pattern vector can only be used once, disallowing that there are different incarnations of the very same anonymization pattern. We have no justification why this should be so and we see no reason to justify this constraint from the viewpoint of data privacy. This leads us to proposing a modified model whose usefulness for practical data anonymization tasks is supported by experiments on real-world data.

Altogether, with our new model we can improve both on k-Anonymity by letting the data user influence the anonymization process as well as on the previous model [4] by allowing the full flexibility for the data user to influence the anonymization process.

Formal Introduction of the New Model. A *row type* is a maximal set of identical rows of a matrix. Matrices are made k-anonymous by suppressing some of their entries. Formally, *suppressing* an entry $M[i, j]$ of an $n \times m$-matrix M over alphabet Σ with $1 \leq i \leq n$ and $1 \leq j \leq m$ means to simply replace $M[i, j] \in \Sigma$ by the new symbol "$\star$", ending up with a matrix over the alphabet $\Sigma \cup \{\star\}$.

Our central enhancement of the k-Anonymity model lies in the user-specific pattern mask guiding the anonymization process: Every row in the k-anonymous output matrix has to conform to one of the given pattern vectors. A row r in a matrix $M \in \{\Sigma, \star\}^{n \times m}$ *matches* a pattern vector $v \in \{\square, \star\}^m$ if and only if $\forall 1 \leq i \leq m : r[i] = \star \iff v[i] = \star$, that is, r and v have $\star$-symbols at the same positions. With these definitions we can now formally define our central computational problem. The decisive difference to our previous model [4] is that in our new model two non-identical output rows can match the same pattern vector.

Pattern-Guided k-Anonymity

Input: A matrix $M \in \Sigma^{n \times m}$, a pattern mask $P \in \{\square, \star\}^{p \times m}$, and two positive integers k and s.

Question: Can one suppress at most s elements of M in order to get a k-anonymous matrix M' such that each row type of M' can be matched to one pattern vector of P?

Our Results. We show that Pattern-Guided k-Anonymity is NP-complete, even if the input matrix only consists of three columns, there are only two pattern vectors, and $k = 3$. Motivated by this computational intractability result, we develop an exact algorithm that solves Pattern-Guided k-Anonymity in $O(2^{tp}t^6p^5m + nm)$ time for an $n \times m$ input matrix M, p pattern vectors, and the number of different rows in M being t. In other words, this shows that Pattern-Guided k-Anonymity is fixed-parameter tractable for the combined parameter (t, p) and actually can be solved in linear time if t and p take constant values. This result paves the way to a formulation of Pattern-Guided k-Anonymity as an integer linear program for exactly solving moderate-size instances of Pattern-Guided k-Anonymity. Furthermore, our fixed-parameter tractability result also leads to a simple and efficient greedy heuristic whose practical competitiveness is underlined by a set of experiments with real-world data, also favorably comparing with the Mondrian algorithm for k-Anonymity [15].

Due to the lack of space, several details and experimental evaluations are deferred to a full version.

2 Complexity and Algorithms

Natural parameters occurring in the problem definition of Pattern-Guided k-Anonymity are the number n of rows, the number m of columns, the alphabet size $|\Sigma|$, the number p of pattern vectors, the degree of anonymity k, and the cost bound s. In general, the number of rows will arguably be large and, thus, also the cost bound s tends to be large. However, analyzing the adult dataset [10] prepared as described by Machanavajjhala et al. [19], it turns out that some of the other mentioned parameters are small: The dataset has $m = 9$ columns and the alphabet size is 73. Furthermore, it is natural to assume that also the number of pattern vectors is not that large. Indeed, compared to the $n = 32,561$ rows even the number of *all possible* pattern vectors $2^9 = 512$ is smaller. Finally there are applications where k, the degree of anonymity, is small [9]. Summarizing, we can state that fixed-parameter tractability with respect to the parameters $|\Sigma|$, m, or p could be of practical relevance. Unfortunately, by reducing from the 3-Set Cover we can show that Pattern-Guided k-Anonymity is NP-hard in very restricted cases.

Theorem 1. Pattern-Guided k-Anonymity *is NP-complete even for two pattern vectors, three columns, and* $k = 3$.

Proof. We reduce from the NP-hard 3-Set Cover [14]: Given a set family $\mathcal{F} = \{S_1, \ldots, S_\alpha\}$ with $|S_i| = 3$ over a universe $U = \{u_1, \ldots, u_\beta\}$ and a positive integer h, the task is to decide whether there is a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most h such that $\bigcup_{S \in \mathcal{F}'} S = U$. In the reduction we need unique entries in the constructed input matrix M. For ease of notation we introduce the $\triangle$-symbol with an unusual semantics. Each occurrence of a $\triangle$-symbol stands for a *different* unique symbol in the alphabet Σ. One could informally state this as "$\triangle \neq \triangle$". We now describe the construction. Let $(\mathcal{F}, U, h)$ be the 3-Set Cover

instance. We construct an equivalent instance (M, P, k, s) of Pattern-Guided k-Anonymity as follows: Initialize M and P as empty matrices. Then, for each element $u_i \in U$ add the row $(u_i, \triangle, \triangle)$ twice to the input matrix M. For each set $S_i \in \mathcal{F}$ with $S_i = \{u_a, u_b, u_c\}$ add to M the three rows (u_a, S_i, S_i), (u_b, S_i, S_i), and (u_c, S_i, S_i). Finally set $k = 3$, $s = 4|U| + 3|\mathcal{F}| + 3h$ and add to P the pattern vectors $(\square, \star, \star)$ and $(\star, \square, \square)$.

We show the correctness of the above construction by proving that $(\mathcal{F}, U, h)$ is a yes-instance of 3-Set Cover if and only if $(M, P, 3, s)$ is a yes-instance of Pattern-Guided k-Anonymity.

"$\Rightarrow$:" If $(\mathcal{F}, U, h)$ is a yes-instance of 3-Set Cover, then there exists a set cover $\mathcal{F}'$ of size at most h. We suppress the following elements in M: First, suppress all $\triangle$-entries in M. This gives $4|U|$ suppressions. Then, for each $S_i \in \mathcal{F}'$ suppress all S_i-entries in M. This gives at most $6|\mathcal{F}'|$ suppressions. Finally, for each $S_j \notin \mathcal{F}'$ suppress the first column of all rows containing the entry S_j. This are $3(|\mathcal{F}| - |\mathcal{F}'|)$ suppressions. Let M' denote the matrix with the suppressed elements. Note that M' contains $4|U| + 3|\mathcal{F}| + 3|\mathcal{F}'| \leq s$ suppressed entries. Furthermore, in each row in M' either the first element is suppressed or the last two elements. Hence, each row of M' matches to one of the two pattern vectors of P. Finally, observe that M' is 3-anonymous: The three rows corresponding to the set $S_j \notin \mathcal{F}'$ are identical: the first column is suppressed and the next two columns contain the symbol S_j. Since $\mathcal{F}'$ is a set cover, there exists for each element u_j a set $S_i \in \mathcal{F}'$ such that $u_j \in S_i$. Thus, by construction, the two rows corresponding to the element u_j and the row (u_j, S_i, S_i) in M coincide in M': The first column contains the entry u_j and the other two columns are suppressed. Finally, for each row (u_i, S_j, S_j) in M that corresponds to a set $S_j \in \mathcal{F}'$ the row in M' coincides with the two rows corresponding to the element u_i: Again, the first column contains the entry u_i and the other two columns are suppressed.

"$\Leftarrow$:" If $(M, P, 3, s)$ is a yes-instance of Pattern-Guided k-Anonymity, then there is a 3-anonymous matrix M' that is obtained from M by suppressing at most s elements and each row of M' matches to one of the two pattern vectors in P. Since M and so M' contain $2|U| + 3|\mathcal{F}|$ rows, M' contains at most $s = 4|U| + 3|\mathcal{F}| + 3h$ suppressions and each pattern vector contains a $\star$-symbol, there are at most $2|U| + 3h$ rows in M' containing two suppressions and at least $3|\mathcal{F}| - 3h$ rows containing one suppression. Furthermore, since the $2|U|$ rows in M corresponding to the elements of U contain the unique symbol $\triangle$ in the last two columns, in M these rows are suppressed in the last two columns. Thus, at most $3h$ rows corresponding to sets of $\mathcal{F}$ have two suppressions in M'. Observe that for each set $S_i \in \mathcal{F}$ the entries in the last two columns of the corresponding rows are S_i. There is no other occurrence of this entry in M. Hence, the at least $3|\mathcal{F}| - 3h$ rows in M' with one suppression correspond to $|\mathcal{F}| - h$ sets in $\mathcal{F}$. Thus, the at most $3h$ rows in M' that correspond to sets of $\mathcal{F}$ and contain two suppressions correspond to at most h sets of $\mathcal{F}$. Denote these h sets by $\mathcal{F}'$. We now show that $\mathcal{F}'$ is a set cover for the 3-Set Cover instance. Assume by contradiction that $\mathcal{F}'$ is no set cover and, hence, there is a set $u \in U \setminus (\bigcup_{S \in \mathcal{F}'} S)$. But since M' is 3-anonymous, there has to be a row r in M' that

corresponds to some set S_i such that this row coincides with the two rows r_1^u and r_2^u corresponding to u. Since all rows in M' corresponding to elements of U contain two suppressions in the last two columns, the row r also contains two suppressions in the last two columns. Thus, $S_i \in \mathcal{F}'$. Furthermore, r has to coincide with r_1^u and r_2^u in the first column, that is, r contains as entry in the first column the symbol u. Hence, $u \in S_i$, a contradiction. □

Contrasting Theorem 1, we will show fixed-parameter tractability with respect to the combined parameter $(|\Sigma|, m)$. To this end, we additionally use as parameter the number t of different input rows. Indeed, we show fixed-parameter tractability with respect to the combined parameter (t, p). This implies fixed-parameter tractability with respect to the combined parameter $(|\Sigma|, m)$ as $|\Sigma|^m \geq t$ and $|\Sigma|^m \geq 2^m \geq p$. This results from an adaption of combinatorial algorithms from previous work [4, 5].

Theorem 2. Pattern-Guided k-Anonymity *can be solved in* $O(2^{tp} \cdot t^6 p^5 \cdot m + nm)$ *time where* p *is the number of pattern vectors and* t *is the number of different rows in the input matrix* M.

ILP Formulation. Next, we describe an integer linear program (ILP) formulation for Pattern-Guided k-Anonymity employing ideas behind the fixed-parameter algorithm of Theorem 2. To this end, we need the following notation. We distinguish between the *input* row types of the input matrix M and the *output* row types of the output matrix M'. Note that in the beginning we can compute the input row types of M in $O(nm)$ time using a trie [11], but the output row types are unknown. By the definition of Pattern-Guided k-Anonymity, each output row type R' has to match a pattern vector $v \in P$. We call R' an *instance* of v.

More specifically, our ILP contains the integer variables $x_{i,j}$ denoting the number of rows from type i being assigned into an output row type compatible with pattern vector j. The binary variable $u_{j,l}$ is 0 if instance l of pattern vector j is used in the solution, that is, there is at least one row mapped to it, otherwise it may be set to 1. Furthermore, n_i denotes the number of rows of type i, ω_j denotes the costs of pattern vector j, and k is the required degree of anonymity. Let $\hat{p}_i \leq t$ denote the number of instances of pattern vector i and let $c(i, j, l)$ be 1 if mapping row i to pattern vector j produces pattern vector instance l, otherwise $c(i, j, l) = 0$. With this notation we can state our ILP formulation:

$$\min \sum_{i=1}^{t} \sum_{j=1}^{p} x(i,j) \cdot \omega_j \tag{1}$$

$$\sum_{i=1}^{t} c(i,j,l) \cdot x_{i,j} \leq (1 - u_{j,l}) \cdot n \qquad \begin{matrix} 1 \leq j \leq p \\ 1 \leq l \leq \hat{p}_j \end{matrix} \tag{2}$$

$$\sum_{i=1}^{t} c(i,j,l) \cdot x_{i,j} + k \cdot u_{j,l} \geq k \qquad \begin{matrix}1 \leq j \leq p \\ 1 \leq l \leq \hat{p}_j\end{matrix} \qquad (3)$$

$$\sum_{j=1}^{p} x_{i,j} = n_i \qquad 1 \leq i \leq t. \qquad (4)$$

The goal function (1) ensures that the solution has a minimum number of suppressions. Constraint (2) ensures that the variables $u_{j,l}$ are consistently set with the variables $x_{i,j}$, that is, if there is some positive variable $x_{i,j}$ indicating that the instance l of pattern vector j is used, then $u_{j,l} = 0$. Constraint (3) ensures that every pattern vector instance that is used by the solution contains at least k rows. Constraint (4) ensures that the solution uses as many rows from each row type as available.

We remark that, as Theorem 2, our ILP formulation also yields fixed-parameter tractability with respect to the combined parameter (t, p). This is due to a famous result of Lenstra [16] and the fact that the number of variables in the ILP is bounded by $O(tp)$. Theorem 2, however, provides a direct *combinatorial* algorithm with better worst-case running time bounds. Nevertheless, in the experimental section we decided to use the ILP formulation and not the combinatorial algorithm based on the experience that there are very strong (commercial) ILP solvers that in practice typically perform much better than the worst-case analysis predicts.

Greedy Heuristic. We now provide a greedy heuristic based on the ideas of the fixed-parameter algorithm of Theorem 2. The fixed-parameter algorithm basically does exhaustive search on the assignment of rows to pattern vectors. More precisely, for each row type R and each pattern vector v it tries both possibilities of whether rows of R are assigned to v or not. In contrast, our greedy heuristic will just pick for each input row type R the "cheapest" pattern vector v and then assigns all compatible rows of M to v. This is realized as follows: We consider all pattern vectors one after the other ordered by increasing number of $\star$-symbols. This ensures that we start with the "cheapest" pattern vector. Then we assign as many rows as possible of M to v: We just consider every instance R' of v and if there are more than k rows in M that are compatible with R', then we assign all compatible rows to R'. Once a row is assigned, it will not be reassigned to any other output row type and, hence, the row will be deleted from M. Overall this gives a running time of $O(pnm)$. See Algorithm 1 for the pseudo-code of the greedy heuristic. If at some point of time there are less than k remaining rows in M, then these rows will be fully suppressed. Note that this slightly deviates from our formal definition of Pattern-Guided k-Anonymity. However, since fully suppressed rows do not reveal any data, this potential violation of the k-anonymity requirement does not matter.

Our greedy heuristic clearly does not always provide optimal solutions. Our experiments indicate, however, that it is very fast and that it typically provides solutions close to the optimum and outperforms the Mondrian algorithm [15] in most datasets we tested. While this demonstrates the practicality of Algorithm 1,

Algorithm 1.. Greedy Heuristic (M, P, k)

```
1: Sort pattern vectors P by cost (increasing order)
2: for each v ∈ P do
3:     Compute all instances of v
4:     for each instance R' of v do
5:         if ≥ k rows are compatible with R' then
6:             Assign all compatible rows of M to R'
7:             Delete the assigned rows from M.
```

the following result shows that from the viewpoint of polynomial-time approximation algorithmics it is weak in the worst case.

Theorem 3. *Algorithm 1 for* Pattern-Guided k-Anonymity *runs in* $O(pnm)$ *time and provides a factor* m*-approximation. This approximation bound is asymptotically tight for Algorithm 1.*

Proof. Since the running time is already discussed above, it remains to show the approximation factor. Let s_{heur} be the number of suppressions in a solution provided by Algorithm 1 and s_{opt} be the number of suppressions in an optimal solution. We show that for every instance it holds that $s_{\text{heur}} \leq m \cdot s_{\text{opt}}$. Let M be a matrix and M'_{heur} be the suppressed matrix produced by Algorithm 1 and M'_{opt} be the suppressed matrix corresponding to an optimal solution. First, observe that if any row of M occurs more than k times, then this row does not contain any suppressed entry in M'_{heur}. Hence, for any row in M'_{opt} not containing any suppressed entry it follows that the corresponding row in M'_{heur} also does not contain any suppression. Clearly, each row in M'_{heur} has at most m entries suppressed. Thus, each row in M'_{heur} has at most m times more suppressed entries than the corresponding row in M'_{opt}.

To show that this upper bound is asymptotically tight, consider the following instance. Set $k = m$ and let M be as follows: The matrix M contains k times the row with the symbol 1 in every entry. Furthermore, for each $i \in \{1, \ldots, m\}$ there are $k - 1$ rows in M such that all but the i^{th} entry contains the symbol 1. In the i^{th} entry each of the $k - 1$ rows contains a uniquely occurring symbol. Finally, the pattern mask simply contains all 2^m possible rows/vectors over the alphabet $\{\square, \star\}$. Algorithm 1 will suppress nothing in the k all-1 rows and will suppress every entry of the remaining rows. This gives $s_{\text{heur}} = (k-1) \cdot m^2 = (m-1) \cdot m^2$ suppressions. However, an optimal solution suppresses in each row exactly one entry: The rows containing in all but the i^{th} entry the symbol 1 are suppressed in the i^{th} entry. Furthermore, to ensure the anonymity requirement, in the submatrix with the k rows containing the symbol 1 in every entry the diagonal is suppressed. Thus, the number of suppressions is equal to the number of rows, that is, $s_{\text{opt}} = k + (k-1)m = m^2$. Hence, $s_{\text{heur}} = (m-1)s_{\text{opt}}$. □

3 Implementation and Experiments

In this section we present the results of our experimental evaluation of the heuristic and the ILP formulation presented in Section 2. We used four datasets for our experimental evaluations; these were taken from the UCI machine learning repository [10]. In this extended abstract we only discuss two of them.
Adult[2]: This was extracted from a dataset of the US Census Bureau Data Extraction System. It consists of 32,561 records over 15 attributes. Since the entries in one attribute are unique for roughly half of the records, we removed this attribute from the dataset. Following Machanavajjhala et al. [19], we prepared also this dataset with nine attributes and call this variant Adult-2.
CMC[3]: This dataset is a subset of the 1987 National Indonesia Contraceptive Prevalence Survey. It contains 1,473 records over 10 attributes.

All our experiments are performed on an Intel Xeon E5-1620 3.6GHz machine with 64GB memory under the Debian GNU/Linux 6.0 operating system. The heuristic is implemented in Haskell. The ILP implementation is using ILOG CPLEX by its C++ API. Both implementations are licensed under GPL Version 3. The source code is available from `http://akt.tu-berlin.de/menue/software/`.

We tested our greedy heuristic in two types of experiments. In the first type we "misused" our greedy heuristic to solve the classical k-Anonymity problem by specifying all possible pattern vectors since we wanted to compare the practical relevance of our greedy heuristic with an existing implementation. We decided to compare with an implementation of the well-known Mondrian [15] algorithm[4] since we could not find a more recent implementation of a k-Anonymity algorithm which is freely available. In the second type of experiments, we analyze the distance of the results provided by our greedy heuristic from an optimal solution (with a minimum number of suppressed entries). The optimal solution is provided by the ILP implementation.

Obvious criteria for the evaluation of the experiments are the number of suppressions and the running time. Furthermore, we use the average size h_{avg} and the maximum size $h_{\max}$ of the output row types as already done by Li et al. [17] and Machanavajjhala et al. [19], as well as the number $\#h$ of output row types. The perhaps most difficult to describe measurement we use is the "usefulness" introduced by Loukides and Shao [18]. Roughly speaking, the *usefulness* is the average tuple diversity of all output row types. Usefulness values lie between zero and the number m of columns. Except for $\#h$, small values indicate better solutions.

Heuristic vs. Mondrian

For each dataset, we computed k-anonymous datasets with our greedy heuristic and Mondrian for $k \in \{2, 3, \ldots, 10, 25, 50, 75, 100\}$. The running time behavior

[2] `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/adult/`
[3] `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/cmc/`
[4] `http://cs.utdallas.edu/dspl/cgi-bin/toolbox/index.php?go=home`

Table 1. Heuristic vs. Mondrian: Results for the Adult dataset. The usefulness is denoted by u, the running time in seconds by r, $\#h$ denotes the number of output row types, h_{avg} denotes the average size of the output row types, and $h_{\max}$ denotes the maximum size of the output row types.

Greedy Heuristic						Mondrian					
k	u	r	$\#h$	h_{avg}	$h_{\max}$	k	u	r	$\#h$	h_{avg}	$h_{\max}$
2	2.062	5.5	14589	2.232	16	2	3.505	2789.4	11136	2.709	61
3	2.290	13.2	9208	3.536	18	3	3.782	1803.5	7306	4.128	61
4	2.470	19.538	6670	4.882	25	4	4.007	1337.860	5432	5.553	61
5	2.615	24.9	5199	6.263	31	5	4.191	1062.0	4325	6.974	61
6	2.738	29.663	4315	7.546	42	6	4.362	885.939	3597	8.385	61
7	2.851	34.126	3669	8.875	53	7	4.498	754.652	3053	9.879	61
8	2.942	37.629	3193	10.198	53	8	4.622	659.184	2663	11.326	61
9	3.026	41.216	2832	11.498	52	9	4.766	588.347	2368	12.737	69
10	3.106	44.8	2559	12.724	56	10	4.875	535.9	2145	14.062	69
25	3.840	79.281	1046	31.129	161	25	6.009	229.248	850	35.485	90
50	4.462	117.0	537	60.635	317	50	6.729	127.4	430	70.144	135
75	4.873	144.536	354	91.980	317	75	7.339	93.621	287	105.094	242
100	5.151	163.582	274	118.836	317	100	7.805	76.005	209	144.316	242

of the tested algorithms is somewhat unexpected. Whereas Mondrian gets faster with increasing k, our greedy heuristic gets faster with decreasing k. The reason why the greedy heuristic is faster for small values of k is that usually the cheap pattern vectors are used and, hence, the number of remaining input rows decreases soon. On the contrary, when k is large, the cheap pattern vectors cannot be used and, hence, the greedy heuristic tests many pattern vectors before it actually starts with removing rows from the input matrix. Thus, for larger values of k the greedy heuristic comes closer to its worst-case running time of $O(pnm)$ with $p = 2^m$.

Adult and Adult-2. Our greedy heuristic anonymized the Adult dataset in less than three minutes for all tested values of k. For $k = 3$ and $k = 4$ Mondrian took more than half an hour to anonymize the dataset. However, in contrast to all other values of k, Mondrian was slightly faster for $k = 75$ and $k = 100$. Except for $h_{\max}$ with $k \geq 25$ all quality measures indicate that our heuristic produces better solutions. The usefulness value of the Mondrian solutions is between 1.5 and 1.7 times the usefulness value of the heuristic for all tested k—this indicates significantly better quality of the results of our heuristic. See Table 1 for details and Figure 1 for an illustration.

The solutions for Adult-2 behave similarly to those for Adult. Our greedy heuristic with a maximum running time of five seconds is significantly faster than Mondrian with a maximum running time of 20 minutes (at least 10 times faster for all tested k). However, the usefulness is quite similar for both algorithms.

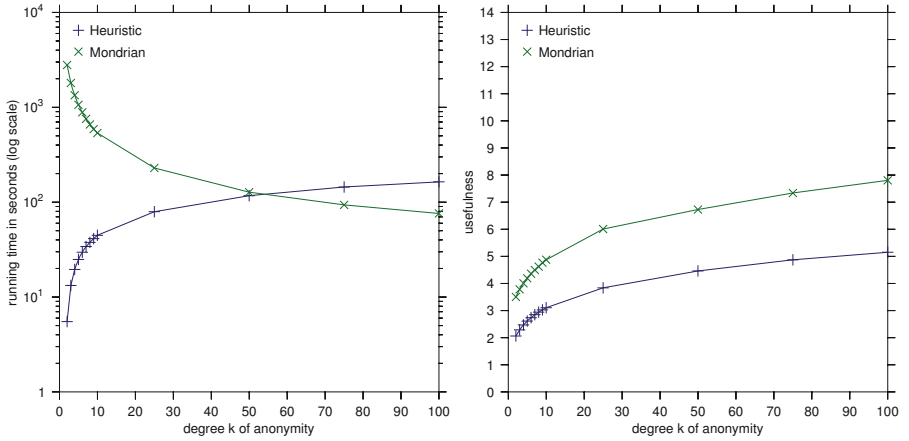


Fig. 1. Comparing running time and usefulness for the Adult dataset

Mondrian beats the heuristic by less than 1% for $k = 50$; the heuristic is slightly better for each other tested k.

CMC. For the CMC dataset, both algorithms were very fast in computing k-anonymous datasets for every tested k. Mondrian took at most 10 seconds and our greedy heuristic took at most 1.2 seconds and was always faster than Mondrian. As for the solution quality, the heuristic can compete with Mondrian. The usefulness of the heuristic results is always slightly better, the Mondrian results have always at least 20% less output row types, and the average output row type size of the heuristic results is always smaller. Only for $k = 5, 6, 7$, and 8, the Mondrian results have a lower maximum size of the output row types.

*Conclusions for Classical k-*Anonymity*. We showed that our greedy heuristic is very efficient even for real-world datasets with more than 30,000 records and with $k \leq 100$. Especially for smaller degrees of anonymity $k \leq 10$, Mondrian is at least ten times slower. Altogether, our heuristic outperforms Mondrian in terms of quality of the solution. Hence, we demonstrated that even for the very special case of specifying *all* possible pattern vectors, our heuristic already produces useful solutions that can at least compete with Mondrian's solutions.

Heuristic vs. Exact Solution

In three scenarios with real-world datasets, we showed that our greedy heuristic performs well in terms of solution quality compared with the optimal solution produced by the ILP implementation. The results of the heuristic are typically within 15% away from the optimal solution to the optimum and in fact for many cases they were optimal, although our heuristic is much more efficient than the

exact algorithm (the ILP was, on average, more than 1000 times slower). The heuristic results tend to get closer to the optimal number of suppressions with increasing degree of anonymity k.

4 Conclusion

We introduced a promising approach to combinatorial data anonymization by enhancing the basic k-Anonymity problem with user-provided "suppression patterns." It seems feasible to extend our model with weights on the attributes, thus making user influence on the anonymization process even more specific. A natural next step is to extend our model by replacing k-Anonymity by more refined data privacy concepts.

On the experimental side, several issues remain to be attacked. For instance, we use integer linear programming in a fairly straightforward way almost without any tuning tricks (e.g., using the heuristic solution or "standard heuristics" for speeding up integer linear program solving). It also remains to perform tests comparing our heuristic algorithm against methods other than Mondrian (unfortunately, for the others no source code seems freely available).

Acknowledgements. We thank Thomas Köhler and Kolja Stahl for their great support in doing implementations and experiments.

References

[1] Blocki, J., Williams, R.: Resolving the complexity of some data privacy problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 393–404. Springer, Heidelberg (2010)

[2] Bonizzoni, P., Della Vedova, G., Dondi, R.: Anonymizing binary and small tables is hard to approximate. Journal of Combinatorial Optimization 22(1), 97–119 (2011)

[3] Bonizzoni, P., Della Vedova, G., Dondi, R., Pirola, Y.: Parameterized complexity of k-anonymity: hardness and tractability. Journal of Combinatorial Optimization (2011)

[4] Bredereck, R., Nichterlein, A., Niedermeier, R., Philip, G.: Pattern-guided data anonymization and clustering. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 182–193. Springer, Heidelberg (2011)

[5] Bredereck, R., Nichterlein, A., Niedermeier, R., Philip, G.: The effect of homogeneity on the computational complexity of combinatorial data anonymization. In: Data Mining and Knowledge Discovery (2012)

[6] Campan, A., Truta, T.M.: Data and structural k-anonymity in social networks. In: Bonchi, F., Ferrari, E., Jiang, W., Malin, B. (eds.) PinKDD 2008. LNCS, vol. 5456, pp. 33–54. Springer, Heidelberg (2009)

[7] Chakaravarthy, V.T., Pandit, V., Sabharwal, Y.: On the complexity of the k-anonymization problem. CoRR, abs/1004.4729 (2010)

[8] Dwork, C.: A firm foundation for private data analysis. Communications of the ACM 54(1), 86–95 (2011)

[9] Evans, P.A., Wareham, T., Chaytor, R.: Fixed-parameter tractability of anonymizing data by suppressing entries. Journal of Combinatorial Optimization 18(4), 362–375 (2009)

[10] Frank, A., Asuncion, A.: UCI machine learning repository(2010), http://archive.ics.uci.edu/ml
[11] Fredkin, E.: Trie memory. Communications of the ACM 3(9), 490–499 (1960)
[12] Fung, B.C.M., Wang, K., Chen, R., Yu, P.S.: Privacy-preserving data publishing: A survey of recent developments. ACM Computing Surveys 14(2) 14:1–14:53 (2010)
[13] Gkoulalas-Divanis, A., Kalnis, P., Verykios, V.S.: Providing k-anonymity in location based services. ACM SIGKDD Explorations Newsletter 12, 3–10 (2010)
[14] Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
[15] LeFevre, K., DeWitt, D., Ramakrishnan, R.: Mondrian multidimensional k-anonymity. In: Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006), pp. 25–25. IEEE (2006)
[16] Lenstra, H.W.: Integer programming with a fixed number of variables. Mathematics of Operations Research 8, 538–548 (1983)
[17] Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007), pp. 106–115. IEEE (2007)
[18] Loukides, G., Shao, J.: Capturing data usefulness and privacy protection in k-anonymisation. In: Proceedings of the 2007 ACM Symposium on Applied Computing, pp. 370–374. ACM (2007)
[19] Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: ℓ-diversity: Privacy beyond k-anonymity. ACM Transactions on Knowledge Discovery from Data 1(1) (2007)
[20] Meyerson, A., Williams, R.: On the complexity of optimal k-anonymity. In: Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2004), pp. 223–228. ACM (2004)
[21] Navarro-Arribas, G., Torra, V., Erola, A., Castellà-Roca, J.: User k-anonymity for privacy preserving data mining of query logs. Information Processing & Management 48(3), 476–487 (2012)
[22] Rastogi, V., Suciu, D., Hong, S.: The boundary between privacy and utility in data publishing. In: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 531–542. VLDB Endowment (2007)

Author Index